UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE POSGRADO

MAESTRIA EN COMPUTACIÓN CON MENCIÓN EN SEGURIDAD INFORMÁTICA

TEMA

"DEVSECOPS CON HERRAMIENTAS OPENSOURCE ORIENTADOS A MICROSERVICIOS PARA LA DETECCIÓN DE VULNERABILIDADES EN EL CICLO DE DESARROLLO DE SOFTWARE DE CORE BANCARIO PARA LA BANCA PERSONAS EN LOS BANCOS PRIVADOS DE LA CIUDAD DE QUITO DEL ECUADOR"

Trabajo de Titulación previo a la obtención del Título de Magíster en Computación con Mención en Seguridad Informática

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética)

AUTORES:

José Alberto Figueroa Rosero

Dora Maricela Pozo Ruiz

Tutor:

José Antonio Quiña Mera, PHD

Asesor:

Xavier Mauricio Rea Peñafiel, MSC

Ibarra, junio de 2025

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	0401421268		
APELLIDOS Y NOMBRES:	Figueroa Rosero José Alberto		
DIRECCIÓN:	Quito, Ricardo Calderón y Vencedores		
EMAIL:	jafigueroar@utn.edu.ec		
TELÉFONO FIJO:	No dispone	**TELÉFONO MÓVIL:**	0989420183

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	0401357355		
APELLIDOS Y NOMBRES:	Pozo Ruiz Dora Maricela		
DIRECCIÓN:	Ibarra, Rio Blanco y Luis Mideros		
EMAIL:	dmpozor1@utn.edu.ec		
TELÉFONO FIJO:	No dispone	**TELÉFONO MÓVIL:**	0982478222

DATOS DE LA OBRA	
TÍTULO:	DEVSECOPS CON HERRAMIENTAS OPENSOURCE ORIENTADOS A MICROSERVICIOS PARA LA DETECCIÓN DE VULNERABILIDADES EN EL CICLO DE DESARROLLO DE SOFTWARE DE CORE

	BANCARIO PARA LA BANCA PERSONAS EN LOS BANCOS PRIVADOS DE LA CIUDAD DE QUITO DEL ECUADOR.
AUTOR (ES):	José Alberto Figueroa Rosero Dora Maricela Pozo Ruiz
FECHA: DD/MM/AAAA	24/03/2025
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	**( ) PREGRADO (X) POSGRADO**
TITULO POR EL QUE OPTA:	Magíster en Computación con Mención en Seguridad Informática
TUTOR:	José Antonio Quiña Mera, PHD
ASESOR:	Xavier Mauricio Rea Peñafiel, MSC

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 4 días del mes de junio de 2025

EL AUTORES:

Firmado electrónicamente por:
JOSE ALBERTO FIGUEROA ROSERO
Validar únicamente con FirmaEC

Nombre: Figueroa Rosero José Alberto

Firmado electrónicamente por:
DORA MARICELA POZO RUIZ
Validar únicamente con FirmaEC

Nombre: Pozo Ruiz Dora Maricela

CERTIFICACIÓN TUTOR Y ASESOR DEL TRABAJO DE TITULACIÓN

Nos permitimos informar que revisado el Trabajo final de Grado " DEVSECOPS CON HERRAMIENTAS OPENSOURCE ORIENTADOS A MICROSERVICIOS PARA LA DETECCIÓN DE VULNERABILIDADES EN EL CICLO DE DESARROLLO DE SOFTWARE DE CORE BANCARIO PARA LA BANCA PERSONAS EN LOS BANCOS PRIVADOS DE LA CIUDAD DE QUITO DEL ECUADOR", de los Maestrantes: José Alberto Figueroa Rosero y Dora Maricela Pozo Ruiz de la Maestría en Computación con Mención en Seguridad Informática, certificamos que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizamos su presentación para los fines legales pertinentes.

Ibarra, a los 24 días del mes de marzo de 2025

	Nombres y Apellidos	**Firma**
Tutor:	José Antonio Quiña Mera, PHD	JOSE ANTONIO QUIÑA MERA Firmado digitalmente por JOSE ANTONIO QUIÑA MERA Nombre de reconocimiento (DN): c=EC, sn=QUIÑA MERA, givenName=JOSE ANTONIO, serialNumber=IDCEC-1002322384, cn=JOSE ANTONIO QUIÑA MERA, 2.5.4.97=TINEC-1002322384001
Asesor:	Xavier Mauricio Rea Peñafiel, MSC	Firmado electrónicamente por: XAVIER MAURICIO REA PEÑAFIEL Validar únicamente con FirmaEC

DEDICATORIA

Este logro académico lo dedicado a DIOS, por brindarme la fortaleza para cumplir esta etapa de mi vida.

A mi compañera de vida, a mis hijos y a mis padres, quienes me han apoyado incondicionalmente a lo largo de mi carrera para terminar esta nueva etapa de mi vida.

Con esta tesis ofrezco un pequeño testimonio de que siempre los tengo presentes en cada paso que doy, y también un recordatorio constante de la importancia del trabajo duro y la educación son pilares fundamentales en nuestras vidas.

José Alberto

A Dios, por darme la oportunidad de vivir esta experiencia enriquecedora y brindarme fortaleza, paciencia y perseverancia necesarias para completar esta meta académica.

A mi familia, especialmente a mis padres y hermanas, por su incondicional apoyo, amor y confianza en mí. Su aliento y ánimo en cada paso del camino ha sido fundamental para seguir adelante incluso en los momentos más difíciles.

A mis profesores, tutor y asesor, quienes con su conocimiento y orientación me guiaron en este camino de aprendizaje, inspirándome a profundizar en el mundo de la seguridad informática.

Finalmente, a todos aquellos que saben sobre la importancia de la seguridad informática es fundamental en el mundo digital actual. Que este trabajo sea un granito de arena para el desarrollo y fortalecimiento de esta disciplina.

Dora Maricela

AGRADECIMIENTO

A Dios, por brindarme la fortaleza, salud y sabiduría necesaria para cumplir las metas que me he propuesto a lo largo de la vida.

Agradezco profundamente a la Universidad Técnica del Norte por su invaluable apoyo en mi desarrollo académico. De igual manera, celebro el profesionalismo, conocimiento y el tiempo dedicado de mi tutor de Tesis.

A mi familia, pilar de nuestro hogar, cuyo amor y apoyo incondicional impulsan mi crecimiento personal y profesional.

José Alberto

Al concluir esta etapa de mi vida académica y profesional, deseo expresar mi más sincero agradecimiento a todas las personas que, de una u otra manera, hicieron posible la culminación de esta tesis.

En primer lugar, a Dios, por darme la fuerza en los momentos difíciles, la claridad en los momentos de duda y la luz para seguir adelante cuando el camino parecía incierto.

A mi familia, especialmente a mis padres y hermanas por su amor incondicional, su apoyo constante y por creer en mí incluso en los momentos más difíciles.

A mi asesor y tutor de tesis, por su guía, paciencia y por compartir su conocimiento en cada etapa de este trabajo. Sus conocimientos y orientación fueron esenciales para el desarrollo de esta investigación.

A mis profesores, por compartir su experiencia y enseñanzas, que han sido la base de mi crecimiento académico y profesional en el campo de la seguridad informática.

A la Universidad Técnica del Norte, por brindarme las herramientas y el espacio necesario para desarrollar mis habilidades en el área de la computación y la seguridad informática.

Finalmente, a los profesionales y expertos en seguridad informática que han inspirado mi interés en esta disciplina, y a quienes trabajan día a día para fortalecer la protección de la información en un mundo cada vez más digitalizado.

Dora Maricela

INDICE DE CONTENIDOS

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN

Las ciberamenazas evolucionan constantemente, y los ataques son cada vez más sofisticados y dirigidos, en un entorno de microservicios, las vulnerabilidades pueden ser explotadas rápidamente, y una brecha en un solo servicio puede comprometer toda la aplicación. En este contexto, surge la necesidad de integrar la seguridad desde las primeras etapas del desarrollo de software en sistemas bancarios, lo que ha dado origen al enfoque DevSecOps. Este modelo busca incorporar prácticas de seguridad de manera continua a lo largo del ciclo de vida del desarrollo y despliegue de aplicaciones, con el objetivo de identificar y mitigar vulnerabilidades de forma proactiva. La presente investigación tiene como propósito implementar un enfoque DevSecOps mediante herramientas de código abierto orientadas a microservicios, con el fin de detectar vulnerabilidades de manera anticipada y garantizar la seguridad integral de las aplicaciones. Los principales hallazgos evidencian que las herramientas más utilizadas por los desarrolladores en este contexto son: GitHub Dashboard (planificación), Git y GitHub (control de versiones), Apache Maven (compilación), SonarQube (pruebas), GitHub Actions (CI/CD), Docker (operaciones), Grafana (monitoreo) y Trivy (Detección de vulnerabilidades). Estas herramientas fueron aplicadas y evaluadas en el desarrollo de un caso práctico de estudio. Los resultados demostraron que la implementación del enfoque DevSecOps permite integrar la seguridad de manera sistemática desde las primeras fases del desarrollo de software bancario. Además, el uso de herramientas de análisis estático, dinámico y escaneo de contenedores permitió identificar vulnerabilidades con antelación, reduciendo los costos de corrección y fortaleciendo la postura de seguridad en cada etapa del ciclo de desarrollo.

Palabras clave: DevSecOps, Seguridad en el desarrollo de software, Microservicios, Análisis de vulnerabilidades, Integración continua y despliegue continuo (CI/CD)

ABSTRACT

Cyber threats are constantly evolving, and attacks are becoming increasingly sophisticated and targeted. In a microservices environment, vulnerabilities can be rapidly exploited, and a breach in a single service may compromise the entire application. In this context, the need to integrate security from the early stages of software development in banking systems has led to the emergence of the DevSecOps approach. This model seeks to continuously incorporate security practices throughout the application development and deployment lifecycle, aiming to proactively identify and mitigate vulnerabilities. The objective of this research is to implement a DevSecOps approach using open-source tools tailored to microservices, enabling early vulnerability detection and ensuring comprehensive application security.

The main findings reveal that the most widely used tools by developers in this context are: GitHub Dashboard (planning), Git and GitHub (version control), Apache Maven (compilation), SonarQube (testing), GitHub Actions (CI/CD), Docker (operations), Grafana (monitoring) and Trivy (vulnerability detection). These tools were applied and evaluated in a practical case study. The results demonstrated that implementing the DevSecOps approach enables the systematic integration of security from the early phases of banking software development. Additionally, the use of static and dynamic analysis tools, along with container scanning, facilitated early vulnerability detection, reducing remediation costs and strengthening security posture at each stage of the development cycle.

Keywords: DevSecOps, Secure software development, Microservices, Vulnerability analysis, Continuous integration and continuous deployment (CI/CD)

CAPITULO I

1 EL PROBLEMA

1.1 Problema de investigación

Las ciberamenazas están en constante evolución, con ataques cada vez más sofisticados y dirigidos. Las vulnerabilidades pueden ser explotadas rápidamente, especialmente en un entorno de microservicios donde una vulnerabilidad en un servicio puede comprometer toda la aplicación debido a la deficiencia en los procesos de control de seguridad en el ciclo de desarrollo de software (Risco y Molina, 2023).

Las instituciones bancarias de la ciudad de Quito, Ecuador manejan grandes volúmenes de datos sensibles y transacciones financieras críticas que las hacen atractivas para los ciberataques. Los sistemas de core bancario, especialmente aquellos diseñados en arquitecturas de microservicios, son vulnerables a problemas de seguridad si no se implementan correctamente mecanismos de protección a lo largo del ciclo de desarrollo. Existe dificultad para implementar un enfoque de seguridad proactivo y automatizado DevSecOps utilizando herramientas open source en un entorno de microservicios, específicamente en el desarrollo de software de core bancario, donde la detección de vulnerabilidades debe ser rápida y eficaz, se enfrenta a desafíos tecnológicos, normativos y operativos. No obstante implementar la cultura DevSecOps en el ciclo de desarrollo de software es un tema complejo y con gran amplitud ya que es un tema que ha tomado fuerza en los últimos años, y al no existir un proceso de estándares internacionales que permita medir una implementación adecuada de DevSecOps no se puede obtener un dato estadístico actualizado sobre el número de empresas que tengan una implementación exitosa.

En este contexto la creciente adopción de arquitecturas de microservicios y la necesidad de integrar la seguridad desde las primeras etapas del desarrollo de software de core bancario, surge el enfoque de DevSecOps, en donde se busca incorporar prácticas de seguridad de manera continua a lo largo del ciclo de vida del desarrollo y despliegue de aplicaciones, con el objetivo de identificar y mitigar vulnerabilidades de forma proactiva. Una de las principales preocupaciones en el desarrollo de microservicios es la detección temprana de vulnerabilidades, dado el alto grado de interdependencia entre los distintos componentes, bajo tal contexto la utilización de herramientas de código abierto se

presenta como una opción atractiva para las organizaciones, ofreciendo flexibilidad, transparencia y, en muchos casos, un costo reducido (Padrón, 2021).

Sin embargo, la implementación efectiva de un enfoque DevSecOps con herramientas de código abierto orientadas a microservicios plantea bastantes desafíos. Por citar uno de ellos radica en la selección adecuada de las herramientas, considerando la diversidad de tecnologías y lenguajes de programación utilizados en el desarrollo de microservicios del backend de un core bancario. Además, es fundamental integrar estas herramientas de manera coherente en los pipelines de desarrollo, asegurando su ejecución automatizada y la generación de alertas en tiempo real.

Otro aspecto crucial es la gestión de falsos positivos y la interpretación de los resultados obtenidos por estas herramientas ya que, dada la complejidad inherente a las arquitecturas de microservicios, es posible que algunas vulnerabilidades reportadas no representen una amenaza real o sean difíciles de explotar en el contexto específico de la aplicación. Por lo tanto, es necesario contar con procesos y procedimientos claros para validar y priorizar las vulnerabilidades detectadas, optimizando así los recursos y el tiempo del equipo de desarrollo (Varela, 2021). También debe considerarse el aspecto cultural y organizativo en la implementación de DevSecOps, todo esto teniendo en cuenta el desarrollo de una cultura de colaboración entre los equipos de desarrollo, operaciones y seguridad, promoviendo siempre la responsabilidad compartida en la gestión de la seguridad del software bancario, ya que una comunicación fluida garantiza la participación activa de todos los involucrados en el proceso y la incorporación de la seguridad como un criterio clave en la toma de decisiones.

1.2 Antecedentes

Con la finalidad de tomar una base referencial de las investigaciones realizadas sobre el tema de estudio, se procedió a establecer la cadena de búsqueda que permita contestar a la siguiente interrogante de investigación

¿Cómo implementar el enfoque de DevSecOps utilizando herramientas de código abierto orientadas a microservicios para la detección proactiva de vulnerabilidades en el ciclo de desarrollo de software, que garantice la seguridad integral de las aplicaciones?

Y se procedió a realizar la consulta mediante el motor de búsqueda Google Académico bajo los siguientes criterios:

- Cadena de búsqueda: "devsecops" and "código abierto" and "microservicios"
- Temporalidad: publicado entre el intervalo de años 2019 y 2024
- Idioma: cualquier idioma

Como resultado de la búsqueda utilizando los criterios mencionados, se obtuvieron un total de 45 resultados como indica la siguiente figura:

Figura 1 Resultados de la cadena de búsqueda realizada en Google Académico

Fuente: (Google Académico, 2024)

Luego de esto, se procedió a revisar el contenido de cada una de las investigaciones encontradas y se tomó en cuenta diversos documentos de índole similar, se seleccionaron un total de siete investigaciones, tales como:

El trabajo de suficiencia profesional “Implementación del modelo DevSecOps en una arquitectura de microservicios on-premise para una empresa del sector bancario” realizado por (Risco y Molina, 2023), el cual se centró en la implementación del modelo DevSecOps en una arquitectura de microservicios on-premise para una empresa bancaria. Todo esto teniendo en cuenta que su objetivo fue agilizar y asegurar los procesos de entrega de software en el contexto de la transformación digital del sector. Se buscaba mejorar la seguridad en todas las etapas del ciclo de desarrollo, según el marco SAMM

v2.0 de OWASP. La iniciativa implicaba un cambio cultural para fomentar la colaboración entre desarrollo, operaciones y seguridad. Otro trabajo considerado fue el trabajo de titulación “Aplicación sobre arquitectura de microservicios en CI desplegada en la nube con seguridad basada en DevSecOps”, elaborado por (Gómez, 2022), mismo el cual se enfocó en una arquitectura de microservicios desplegada en la nube, impulsada por principios de integración continua y desarrollo continuo, destacándose la eficiencia de dividir servicios en microservicios para mejorar la comunicación entre aplicaciones. Así se priorizó la seguridad mediante métodos avanzados, especialmente en la comunicación con APIs, bajo la filosofía de DevSecOps. Todo esto partiendo de la base de que se promovieron los beneficios de la nube y se proporcionaría una guía para implementar la arquitectura propuesta y segurizar las aplicaciones. Otro trabajo considerado fue la tesis de grado titulada “Diseño y desarrollo de una arquitectura de Internet de las cosas de nueva generación orientada al cálculo y predicción de índices compuestos aplicada en entornos reales” escrita por (Lacalle, 2022), el cual abordó el diseño y desarrollo de una arquitectura IoT de nueva generación para calcular y predecir índices compuestos en entornos reales. Para ello se aprovecharon avances como la IA y el edge computing, contexto bajo el cual se propuso una arquitectura flexible y escalable, con énfasis en la automatización de la generación de indicadores compuestos en tiempo real. Finalmente, cabe destacar que se incluyó el diseño e integración de módulos para adquisición de datos, procesamiento, visualización y seguridad. También el trabajo de titulación “Segurización de microservicios desplegados con Docker y orquestados con Kubernetes”, elaborado por (Rhazili, 2023), el cual se centró en la Segurización de microservicios desplegados mediante Docker y orquestados con Kubernetes. Destacando la migración hacia esta arquitectura por sus ventajas, como la escalabilidad y la independencia funcional. Motivo por el cual se resalta el aumento en la adopción de estas tecnologías, lo que los convierte en objetivos atractivos para los ciberdelincuentes. Todo esto debido a que el proyecto enfocó en analizar la seguridad de los microservicios de Lansweeper, tanto en entornos locales como en la nube, e implementar mecanismos de protección.

Se consideró también el articulo investigativo “DevSecOps: integración de la seguridad en entornos CI/CD”, realizado por (Padrón, 2021), dentro del cual se enfocó en la integración de la seguridad en entornos CI/CD mediante la cultura DevSecOps. Reconociendo la necesidad de automatizar la seguridad sin comprometer la agilidad del desarrollo, se investigaron servidores de integración continua y herramientas de

automatización de seguridad. GitHub Actions se empleó para desarrollar un pipeline que incluyera diversas pruebas de seguridad. Aunque el trabajo tuvo limitaciones presupuestarias y de prueba en entornos reales, los resultados proporcionaron una comprensión valiosa de DevSecOps en el ámbito empresarial. Se tomo en cuenta también la tesis de grado "Desarrollo seguro de software bajo la metodología DevSecOps", elaborada por (Varela, 2021), cuya investigación se centró en el desarrollo seguro de software bajo la metodología DevSecOps, bajo el cual se exploró cómo la seguridad de la información ha influenciado el diseño de aplicaciones, adoptando el enfoque del security by design. DevSecOps se destacó como una metodología ágil que integra la seguridad en el ciclo de vida de las aplicaciones. Finalmente, se tomó como precedente el trabajo de titulación "Creación de una aplicación Backend del módulo de transferencias (cuentas y montos), con microservicios Graphql y Rest en contenedores Docker, para fomentar el levantamiento de una arquitectura Devops", elaborado por (Guitarra, 2022), el cual se enfocó en la creación de una aplicación Backend para el módulo de transferencias, utilizando microservicios GraphQL y REST en contenedores Docker para promover una arquitectura DevOps. Abordando problemas comunes en el despliegue de aplicaciones al integrar nuevos servicios. En donde se implementaron tecnologías como Docker y microservicios para mejorar la eficiencia de las aplicaciones, motivo por el cual se realizó una experimentación computacional basada en la guía Wholin para comparar el rendimiento entre la arquitectura Docker y el entorno Localhost.

En base a las investigaciones citadas se puede mencionar que los trabajos de estudio realizados, aunque comparten temas comunes como DevSecOps, microservicios y seguridad, se distinguen por sus enfoques específicos, el contexto en el que se aplican, y las tecnologías que utilizan, mientras algunos se centran en arquitecturas específicas (on-premise, nube, IoT), otros toman la seguridad desde diferentes ángulos (integración en CI/CD, segurización de microservicios). Esto muestra la diversidad y complejidad de los desafíos y soluciones en el desarrollo de software seguro. Así como también se procedió a realizar la consulta en la base de datos científica IEEE Xplore bajo los siguientes criterios:

- Cadena de búsqueda: (("All Metadata": devsecops) AND ("All Metadata": microservices))
- Temporalidad: publicado entre el intervalo de años 2019 y 2024

Como resultado de la búsqueda utilizando los criterios mencionados, se obtuvieron un total de trece resultados entre conferencias y libros, como muestra la figura 2

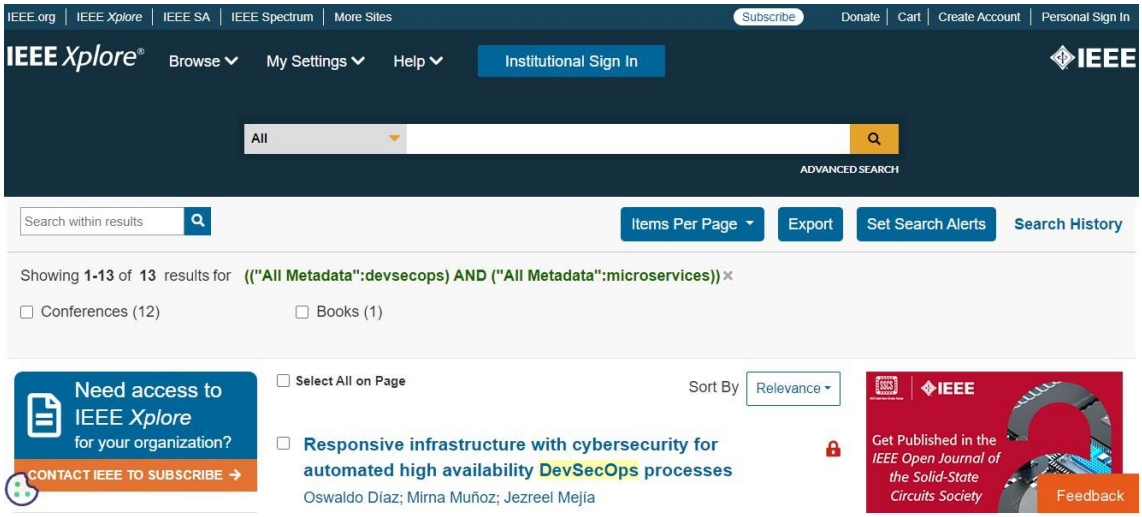


Figura 2 Resultados de la cadena de búsqueda realizada en IEEE Xplore

Fuente: (IEEE Xplore, 2024)

A continuación, se procedió a revisar el contenido de cada una de las investigaciones encontradas y se tomó en cuenta documentos similares al tema de estudio y se seleccionaron un total de cinco investigaciones que se detallan a continuación:

La conferencia " Monitoring solution for cloud-native DevSecOps", elaborada por (Sojan y otros, 2021). Este estudio presenta una solución basada en microservicios que monitorea la infraestructura como el nivel de aplicación y añade automatización para facilitar el despliegue y la alerta por eventos, el desarrollo y las operaciones de software están adoptando cada vez más entornos nativos en la nube, impulsados por prácticas como DevSecOps y la monitorización es esencial en DevSecOps.

El artículo "A Unified Framework for Automating Software Security Analysis in DevSecOps" elaborado por (Aljohani y Alqahtani, 2023). Este artículo describe los desafíos actuales en la seguridad de aplicaciones DevOps, como la falta de herramientas automatizadas, la integración insuficiente de herramientas de seguridad, la falta de conocimiento en seguridad, y los falsos positivos de los escáneres de vulnerabilidades. Se propone un marco unificado para automatizar el análisis de seguridad en DevSecOps y se presenta un caso de estudio para demostrar su aplicabilidad.

El trabajo de investigación “Modeling and Analysis of Dependencies between Microservices in DevSecOps”, realizado por (McZara y otros, 2020). Esta investigación utiliza la matriz de estructura de diseño (DSM) para modelar y analizar los enlaces críticos, apoyando la resiliencia de las aplicaciones basadas en microservicios en DevSecOps, estas aplicaciones constan de microservicios independientes, cada uno con sus propios objetivos y recursos, una interrupción puede afectar a varios microservicios, la recuperación rápida de aplicaciones críticas depende de identificar y priorizar las dependencias entre microservicios.

El capítulo del libro “Securing Modern Applications and Systems” elaborado por (Janca, 2021). Este capítulo explica técnicas de seguridad para APIs y microservicios, almacenamiento en línea, contenedores, orquestación, flujos de trabajo en la nube, serverless, infraestructura como código, seguridad como código, PaaS, IaaS, CI/CD, DevSecOps. Analiza herramientas de seguridad diseñadas para integrarse en pipelines de CI/CD, destacando la importancia de proteger las infraestructuras en entornos de desarrollo continuo.

La conferencia “Amazon Web Services Cloud Compliance Automation with Open Policy Agent” elaborada por (Paul y otros, 2024). Esta investigación ayuda a las organizaciones a mejorar la seguridad de sus sistemas en la nube al identificar y corregir problemas proactivamente, propone un enfoque DevSecOps para la seguridad de IaC en entornos en la nube, integrando validación y remediación continua en el pipeline de CI/CD, utilizando Open Policy Agent (OPA) para evaluar plantillas de CloudFormation donde cualquier falta de políticas de seguridad puede detener el despliegue.

Finalmente, luego de la revisión de la literatura en la base de datos científica IEEE Xplore, se menciona que las investigaciones resaltan la importancia de DevSecOps en la protección de infraestructuras y aplicaciones seguras. Se proponen soluciones innovadoras para la monitorización con microservicios, la automatización del análisis de seguridad, y la priorización de dependencias críticas, además se resalta la integración de herramientas de seguridad en pipelines CI/CD y el uso de Open Policy Agent (OPA) para asegurar la seguridad continua en entornos en la nube. Estas estrategias son fundamentales para mejorar la seguridad y la eficiencia en el desarrollo de software seguro.

1.3 Objetivos de la investigación

1.3.1 Objetivo general

Implementar un enfoque de DevSecOps utilizando herramientas de código abierto orientadas a microservicios para la detección proactiva de vulnerabilidades en el ciclo de desarrollo de software, que garantice la seguridad integral de las aplicaciones.

1.3.2 Objetivos específicos

Integrar herramientas de seguridad de código abierto diseñadas para microservicios en los pipelines de desarrollo, asegurando su ejecución automatizada en cada etapa del ciclo de vida del software, mediante la definición de un caso de estudio.

Establecer procesos efectivos para la gestión de vulnerabilidades detectadas, incluyendo la validación, priorización y mitigación de riesgos, con el objetivo de optimizar los recursos y el tiempo del equipo de desarrollo.

Analizar el nivel de cumplimiento de la aplicación de DevSecOps de acuerdo a la norma ISO/IEC 27034 en el proceso de desarrollo de software.

Validar el enfoque DevSecOps determinado métricas de seguridad durante el ciclo de desarrollo del software del caso de estudio seleccionado.

Definir un plan de integración para el desarrollo, seguridad y operaciones en el ciclo de desarrollo de software utilizando herramientas open source.

1.4 Justificación

Dentro del panorama actual del desarrollo de software, la adopción de arquitecturas de microservicios ha ganado una amplia aceptación debido a su capacidad para mejorar la escalabilidad, la flexibilidad y la eficiencia en el desarrollo y despliegue de aplicaciones. Sin embargo, esta evolución también ha planteado desafíos significativos en términos de seguridad, esto debido a que la naturaleza distribuida y altamente interconectada de los microservicios puede crear puntos de vulnerabilidad adicionales, aumentando la exposición a posibles ataques cibernéticos (Padrón, 2021). Los bancos manejan datos extremadamente sensibles y grandes volúmenes de transacciones financieras que

requieren altos niveles de protección. En un contexto de ciberseguridad cada vez más complejo, las instituciones financieras, particularmente los bancos privados, son uno de los principales objetivos de los ataques cibernéticos, como el robo de datos, fraudes y otras amenazas. El uso de DevSecOps integrado con herramientas de código abierto para mejorar la seguridad del software bancario resulta indispensable, ya que permite identificar y mitigar vulnerabilidades de forma proactiva a lo largo del ciclo de vida del desarrollo.

En respuesta a esta creciente preocupación, el enfoque de DevSecOps ha surgido como una estrategia integral para integrar la seguridad desde las primeras etapas del ciclo de vida del desarrollo de software de core bancario ya que, al combinar prácticas de desarrollo ágil con medidas de seguridad proactivas, DevSecOps busca garantizar que la seguridad sea una consideración fundamental en cada fase del proceso de desarrollo y despliegue de aplicaciones.

La incorporación de DevSecOps en el ciclo de desarrollo de software no solo mejorara la seguridad, sino que también optimizara los tiempos de desarrollo y despliegue de aplicaciones. En lugar de considerar la seguridad como un elemento secundario o añadido al final del proceso, DevSecOps integra la seguridad desde el inicio, lo que disminuye los costos y esfuerzos asociados con la corrección de vulnerabilidades detectadas en etapas avanzadas del desarrollo. Esto es relevante en el desarrollo de software de core bancario, donde la banca personas depende de plataformas estables y seguras para gestionar las finanzas de los clientes.

Una parte fundamental de la implementación exitosa de DevSecOps en entornos de microservicios es la elección adecuada de herramientas. Aquí es donde las soluciones de código abierto desempeñan un papel crucial. Tales herramientas ofrecen una variedad de ventajas, como la transparencia, la flexibilidad y, en muchos casos, un costo reducido en comparación con las soluciones propietarias. Además, al ser de código abierto, permiten una mayor colaboración y contribución de la comunidad, lo que puede traducirse en una mayor calidad y seguridad del software (Risco y Molina, 2023).

En el contexto específico de la detección de vulnerabilidades, las herramientas de código abierto orientadas a microservicios ofrecen una amplia gama de funcionalidades, desde análisis estático de código hasta pruebas de penetración automatizadas. Puesto que tales herramientas pueden integrarse fácilmente en los pipelines de desarrollo, permitiendo la

identificación temprana de vulnerabilidades y su corrección antes de que se conviertan en problemas de seguridad más graves en producción. Además, el uso de herramientas de código abierto en un enfoque DevSecOps fomentara una cultura de transparencia o colaboración en toda la organización. Todo esto teniendo en cuenta que al permitir que los equipos de desarrollo, operaciones y seguridad trabajen juntos en la identificación y mitigación de riesgos, promoverá una mayor conciencia y responsabilidad compartida en materia de seguridad del software de core bancario. Lo cual no solo mejorara la calidad y la seguridad del producto final, sino que también fortalece la cultura organizativa en su conjunto.

El estudio también contribuirá al campo de DevSecOps aplicado a entornos bancarios, lo que no ha sido ampliamente explorado en el contexto local de Ecuador. A nivel práctico, la investigación proporcionará un marco que puede ser utilizado por otras instituciones financieras en la región para mejorar su infraestructura tecnológica y seguridad.

En aportes al área de conocimiento, la implementación de DevSecOps y tecnologías basadas en microservicios podría ser una base en la estrategia de transformación digital del país. Esto facilitara la creación de servicios digitales más seguros, ágiles y escalables, mejorando así la calidad del software bancario ofrecido a los clientes.

En cuanto al desarrollo regional o local, al fomentar el uso de herramientas de código abierto y DevSecOps, se impulsará el desarrollo de software financiero nacional, generando empleo cualificado y reduciendo la dependencia de soluciones extranjeras. Además, favorecerá al logro de políticas u objetivos globales, nacionales o regionales, los resultados de esta investigación podrán servir como base para la formulación de políticas públicas que promuevan la adopción de DevSecOps y herramientas de código abierto en el sector de desarrollo de software financiero permitiendo el desarrollo de aplicaciones y servicios digitales más seguros y confiables, esto resultara una mayor eficiencia operativa en el proceso de desarrollo del software de core bancario.

Así como también contribuirá al mejoramiento de la calidad de vida de una comunidad o grupo humano ya que la investigación podrá fomentar la creación de comunidades de desarrolladores y profesionales de la seguridad informática, donde se compartan conocimientos y experiencias, para esto es necesario implementar programas de capacitación para profesionales del sector, con el objetivo de difundir las mejores prácticas y herramientas en este campo.

Para finalizar, se apoya al mejoramiento de una técnica o proceso debido a que el uso de microservicios y herramientas de código abierto fomentara la innovación, ya que estas tecnologías permiten una mayor flexibilidad y escalabilidad en el desarrollo de software así como la integración de DevSecOps en el desarrollo de software bancario puede mejorar la seguridad de las aplicaciones y servicios digitales de la banca personas, esto es primordial para proteger la infraestructura y la información de ataques cibernéticos lo que puede llevar a soluciones de seguridad más robustas y adaptables.

Con el tema propuesto de tesis la Universidad Técnica del Norte UTN podrá tener reconocimiento internacional ya que aplicara técnicas de seguridad de DevSecOps, en el proceso de desarrollo de software que son aplicadas en empresas internacionales de desarrollo de software financiero. En el marco del Plan de Desarrollo para el Nuevo Ecuador 2024-2025, un aspecto fundamental es el aporte del tema de investigación "DevSecOps con herramientas opensource orientadas a microservicios para la detección de vulnerabilidades en el ciclo de desarrollo de software de core bancario para la banca personas en los bancos privados de la ciudad de Quito, Ecuador", podrá mejorar la eficiencia del desarrollo de software, la detección temprana de vulnerabilidades para proteger los sistemas informáticos financieros del país y garantizar la seguridad de la información, este tema se alinea directamente con los objetivos de ciberseguridad del plan de desarrollo, sino que también la adopción de DevSecOps y herramientas de código abierto promoverá la innovación tecnológica, un pilar fundamental en el plan de desarrollo, al utilizar estas prácticas, nuestro país podrá posicionarse como líder en el desarrollo de software seguro y de alta calidad, el desarrollo de competencias locales y la soberanía tecnológica; aspectos son fundamentales para el desarrollo sostenible y la modernización del país (Secretaría Nacional de Planificación , 2024).

La línea de investigación en la cual se basará el proyecto es la seguridad de la información mediante la aplicación de la detección temprana de vulnerabilidades en el ciclo de desarrollo de software, con el fin de garantizar la seguridad de la información de los sistemas informáticos financieros.

CAPITULO II

2 MARCO REFERENCIAL

2.1 Antecedentes

En los últimos años, la adopción de arquitecturas de microservicios ha transformado el desarrollo de aplicaciones, permitiendo una mayor escalabilidad y flexibilidad en la entrega de productos de software. Esta evolución técnica, sin embargo, conlleva desafíos significativos en términos de seguridad, pues cada microservicio opera como un componente independiente y puede convertirse en un punto potencial de vulnerabilidad. Dado el aumento en la complejidad de la infraestructura y en la cantidad de puntos de entrada, se hace fundamental fortalecer la seguridad desde etapas tempranas del desarrollo (Mahalakshmi y Magesh, 2020).

Para mitigar estos riesgos, la integración de prácticas de seguridad desde las primeras fases del ciclo de vida del software se ha vuelto esencial; en este contexto, el enfoque DevSecOps ha ganado prominencia, promoviendo la colaboración entre los equipos de desarrollo, seguridad y operaciones, de forma que la seguridad se integre a lo largo de todo el proceso de construcción y despliegue de aplicaciones. Este enfoque holístico busca automatizar las pruebas de seguridad, el escaneo de vulnerabilidades y la aplicación de parches, reduciendo el tiempo de respuesta ante posibles amenazas y reforzando la resiliencia de las aplicaciones (Rahman y Williams, 2019).

De igual manera, la adopción de metodologías ágiles, combinada con la filosofía DevSecOps, ha demostrado mejorar la calidad del software y acortar los tiempos de entrega, sin perder de vista las demandas de seguridad que exigen sectores como el financiero, el de telecomunicaciones o el gubernamental. Estudios recientes revelan que una implementación correcta de DevSecOps, con la colaboración activa de todos los equipos involucrados, incrementa significativamente la eficiencia en la detección de vulnerabilidades y reduce los costos asociados a la corrección de errores de seguridad en etapas posteriores del desarrollo (Biehl et al., 2021).

El uso de contenedores y plataformas de orquestación como Kubernetes ha sido uno de los catalizadores de la arquitectura de microservicios. No obstante, este mismo entorno de alta disponibilidad y escalabilidad introduce nuevos retos de seguridad, pues cada contenedor y pod puede exponer vulnerabilidades particulares. La filosofía DevSecOps

aporta prácticas de “seguridad como código”, donde las configuraciones de red, políticas de acceso y gestión de secretos son definidas e implementadas de forma automatizada, reduciendo la dependencia de configuraciones manuales que podrían generar brechas de seguridad (Ma et al., 2020). En este sentido, las herramientas de código abierto desempeñan un papel crucial en la adopción de prácticas DevSecOps, ya que ofrecen flexibilidad y permiten a las organizaciones personalizar sus estrategias de seguridad según sus necesidades específicas, sin incurrir en costos prohibitivos por licencias. Un ejemplo claro de ello es Trivy, un escáner de vulnerabilidades de código abierto diseñado para detectar fallas en paquetes a nivel de sistema operativo, así como en archivos de dependencias de diversas aplicaciones. Su naturaleza abierta posibilita la colaboración de la comunidad y la mejora continua de sus capacidades, convirtiéndose en una herramienta esencial para equipos de DevOps y seguridad al integrarla en flujos de trabajo de integración y entrega continua. Además de Trivy, existen otras soluciones como OWASP ZAP, SonarQube (versión Community), Clair o Anchore, que permiten escanear contenedores y aplicaciones en busca de vulnerabilidades (Aqua Security, 2023).

El sector empresarial enfrenta constantemente cambios regulatorios y estrictos controles de cumplimiento, sobre todo en industrias como la financiera y gubernamental. En este contexto, la implementación de DevSecOps permite no solo una integración temprana de la seguridad, sino también la automatización de auditorías y la generación de evidencias que respalden el cumplimiento de normativas. De acuerdo con Ventura el at. (2023), la adopción de un enfoque integrado de seguridad en entornos con microservicios facilita la trazabilidad de componentes y la identificación de riesgos potenciales, resultando en una mayor eficiencia para responder a requisitos regulatorios como la Ley de Protección de Datos Personales o marcos específicos como la ISO/IEC 27034.

Uno de los principales factores que impulsan o frenan la adopción de DevSecOps es la cultura organizacional. La transformación cultural implica romper silos tradicionales y promover equipos multidisciplinarios que comprendan y adopten prácticas de seguridad de manera colaborativa. Esto requiere entrenar al personal en buenas prácticas de codificación segura, herramientas de análisis estático y dinámico, así como inculcar la mentalidad de “seguridad por diseño” en todas las etapas del desarrollo. Según estos autores, las empresas que han logrado una verdadera integración cultural de DevSecOps reportan disminuciones significativas en incidentes críticos de seguridad y tiempos de remediación (Erich et al., 2017).

La consolidación de un enfoque DevSecOps y la utilización de herramientas de código abierto también contribuyen a la formación de una cultura de seguridad dentro de las organizaciones. Al fomentar la responsabilidad compartida y la colaboración entre los diferentes equipos involucrados en el desarrollo, prueba, despliegue y operación de aplicaciones, se refuerza el compromiso con la protección de datos y la integridad de los sistemas. Esta cultura de seguridad adquiere especial relevancia frente a amenazas emergentes en el panorama actual de ciberseguridad y en sectores con altos requerimientos de cumplimiento normativo, como el financiero. (Banusevicute, Jusas y Rojus, 2022).

Otro elemento fundamental en la adopción de DevSecOps es la observabilidad a lo largo de todo el ciclo de vida de desarrollo y operación. Al combinar monitorización, logging y trazabilidad de eventos en tiempo real, las organizaciones pueden detectar anomalías y ataques potenciales de manera temprana. Herramientas de código abierto para análisis de logs y métricas, como Elastic Stack o Prometheus, se pueden enlazar con pipelines de CI/CD para establecer alertas automáticas e informes de seguridad. La automatización continua de estas tareas no solo acelera la respuesta ante amenazas, sino que también libera a los equipos de procesos repetitivos, permitiéndoles enfocarse en la mejora constante de la aplicación y la infraestructura (Stenzel et al., 2021).

Finalmente, el marco regulatorio y los estándares internacionales, como la ISO/IEC 27034 orientada a la seguridad en el ciclo de vida del desarrollo de software, así como guías de OWASP (Open Web Application Security Project) sobre vulnerabilidades más comunes, se han convertido en referencias fundamentales para alinear las buenas prácticas de DevSecOps con los controles de seguridad establecidos. De esta forma, se logra un equilibrio entre la agilidad en la entrega de software y el cumplimiento de requisitos legales y normativos. En consecuencia, la creciente aceptación de DevSecOps no solo responde a necesidades operativas, sino que refleja la evolución de la mentalidad organizacional hacia la responsabilidad compartida y la prevención proactiva de riesgos cibernéticos.

2.2 Marco Teórico

En la actualidad, el desarrollo de software enfrenta un entorno de constantes amenazas y requisitos de cumplimiento cada vez más estrictos, especialmente en sectores críticos

como el bancario. Las prácticas de integración y entrega continua han evolucionado hacia un enfoque más holístico denominado DevSecOps, en el que la seguridad se integra de manera temprana y constante en todo el ciclo de vida del desarrollo (Rahman y Williams, 2019). Este paradigma responde a la necesidad de enfrentar ataques cada vez más sofisticados, al mismo tiempo que se mantiene la agilidad y la eficiencia en la entrega de aplicaciones y servicios.

La transformación digital en el sector bancario ha impulsado la adopción de arquitecturas de microservicios y metodologías ágiles para responder con rapidez a las demandas del mercado. Sin embargo, este entorno caracterizado por la distribución de servicios y la manipulación de datos sensibles, se intensifica la necesidad de mecanismos de seguridad que garanticen la confidencialidad, integridad y disponibilidad de la información financiera (Shahin et al., 2019). Es en este escenario donde surge la filosofía DevSecOps, un enfoque que integra a los equipos de desarrollo, seguridad y operaciones para automatizar y mover a la izquierda los controles de seguridad en todas las fases del ciclo de vida del software (Rajapakse et al., 2021).

En este capítulo, se exponen los conceptos y teorías fundamentales que enmarcan la presente investigación, en primer lugar, se describen las vulnerabilidades y amenazas más frecuentes, luego, se profundiza en la filosofía de DevSecOps y la arquitectura de microservicios, destacando su sinergia con contenedores y orquestadores. Además, se abordarán las normativas y requerimientos específicos del sector bancario y, finalmente, se presentarán las herramientas clave que facilitan la integración de la seguridad en cada etapa del ciclo de vida del software.

2.2.1 Vulnerabilidades y Amenazas en el Desarrollo de Software

Las amenazas a la seguridad del software han evolucionado de manera acelerada, impulsadas por la creciente complejidad de las aplicaciones y la proliferación de servicios en la nube. En entornos distribuidos o basados en microservicios, cada componente puede transformarse en un vector de ataque si no se aplican controles adecuados de autenticación, autorización y cifrado de datos; a esto se suma la continua aparición de vulnerabilidades en bibliotecas externas y frameworks, que se utilizan para agilizar el desarrollo, pero pueden exponer a la organización a fallos críticos si no se gestionan correctamente. Tales deficiencias son más críticas en sectores altamente regulados, como

el bancario, donde la pérdida de datos o la interrupción de servicios se traduce en un impacto financiero y reputacional significativo (Islam y Akter, 2021).

Dentro de las vulnerabilidades más frecuentes se hallan las inyecciones (por ejemplo, SQL Injection o Command Injection), la exposición de datos sensibles (mediante configuraciones erróneas o conexiones inseguras) y las dependencias vulnerables que, al no actualizarse oportunamente, dejan el sistema abierto a exploits conocidos. Además, la arquitectura basada en contenedores (Docker) y orquestadores (Kubernetes) introduce nuevos desafíos, como la administración de secretos, la configuración de permisos de red, y la ejecución de pods con privilegios innecesarios. Estas brechas se exacerban cuando los equipos de desarrollo trabajan con prisa y no incorporan prácticas de seguridad desde la fase inicial (Ma et al., 2020).

La necesidad de abordar dichas amenazas de manera proactiva y sistemática surge del riesgo de interrupciones costosas y violaciones a la confidencialidad de los datos que pueden derivar en sanciones por incumplimiento normativo (Ventura et al., 2021). Además, estudios recientes evidencian que la automatización de pruebas de seguridad y la inclusión de la seguridad en cada fase del desarrollo que son la esencia de enfoques como DevSecOps, reducen drásticamente la exposición a incidentes y el costo de corrección de vulnerabilidades (Islam y Akter, 2021). Por ello, se tiende a plantear soluciones y metodologías que buscan integrar la seguridad de manera temprana y continua, con especial énfasis en los enfoques DevSecOps y las herramientas de código abierto.

2.2.1.1. Panorama Actual de Amenazas.

El crecimiento exponencial de aplicaciones y servicios en línea ha abierto un abanico de oportunidades para organizaciones y usuarios, al tiempo que incrementa la superficie de ataque disponible para ciberdelincuentes. La adopción masiva de servicios en la nube, la virtualización y los contenedores ha promovido la entrega continua de software, lo cual permite a las empresas responder más rápidamente a las necesidades del mercado, sin embargo, esta agilidad conlleva el riesgo de introducir vulnerabilidades en etapas tempranas del desarrollo si los controles de seguridad no se integran de manera sistemática. En consecuencia, muchas aplicaciones web y APIs se exponen de forma

prematura, quedando susceptibles a ataques como inyecciones, accesos no autorizados o escaladas de privilegios (Zhang et al., 2022).

Por otra parte, los ataques cada vez más dirigidos y sofisticados se centran en la capa aplicación, explotando fallas específicas en la lógica de negocio, la validación de entradas y la comunicación entre servicios, este tipo de ofensivas se ve reforzado por técnicas como el reconocimiento avanzado de la infraestructura (fingerprinting) y el uso de herramientas automatizadas para descubrir dependencias vulnerables. Aunado a esto, la naturaleza distribuida de las arquitecturas modernas, que incluyen a los microservicios, complica la identificación temprana de amenazas. Si no se realiza un monitoreo continuo y detallado, los adversarios pueden aprovechar brechas minúsculas para comprometer componentes y, con ello, la totalidad del sistema y de ahí la importancia de fortalecer y automatizar las pruebas de seguridad en cada fase del ciclo de vida del software (Ghali y Guban, 2021).

2.2.1.2. Principales Vulnerabilidades en Arquitecturas Modernas.

La creciente complejidad de las aplicaciones actuales, fundamentadas en microservicios y contenedores, expone a las organizaciones a un amplio espectro de vulnerabilidades que pueden comprometer la confidencialidad y disponibilidad de los datos. Entre las fallas más comunes se destacan la inyección de código, la exposición de datos sensibles, fallas en los controles de acceso y las vulnerabilidades de configuración, todas ellas susceptibles de ser explotadas para escalar privilegios o extraer información confidencial (OWASP Foundation, 2021).

En el contexto de contenedores y microservicios, las amenazas se agravan por la interdependencia entre múltiples servicios, los errores de configuración pueden facilitar la escalada de permisos y la propagación de un ataque dentro de la misma infraestructura (Ma et al., 2020), asimismo, la comunicación interna entre microservicios, si no se asegura mediante protocolos cifrados y una adecuada segmentación de redes, puede servir como vía para que un atacante se desplace lateralmente a otros servicios. Esto resulta especialmente crítico ya que de ser manipulados maliciosamente se puede impactar en la lógica de negocio o en los mecanismos de autenticación y autorización (Zhang et al., 2022).

Frente a este panorama, la adopción de controles de seguridad adecuados en la capa aplicación es fundamental para prevenir el abuso de vulnerabilidades. Prácticas como la validación exhaustiva de las entradas, la gestión segura de dependencias, la autenticación reforzada y la gestión centralizada de secretos son estrategias efectivas para reducir la superficie de ataque y contener posibles incidentes. En este sentido, metodologías como DevSecOps contemplan la automatización de pruebas de seguridad (SAST/DAST), el monitoreo continuo de integridad y el análisis constante de imágenes de contenedor, con el objetivo de detectar y corregir fallos en etapas tempranas del ciclo de vida del software.

2.2.1.3. Impacto en el Sector Bancario.

El sector bancario se caracteriza por manejar información altamente sensible, como datos de cuentas, transacciones financieras y perfiles crediticios de los clientes. Esta sensibilidad de los datos hace que cualquier brecha de seguridad posea un potencial disruptivo excepcionalmente alto, tanto en términos económicos como reputacionales (Mohan & Yapa, 2019). La pérdida de confianza del público y de los inversionistas puede generar efectos a largo plazo, traducidos en una disminución de la competitividad e, incluso, en sanciones regulatorias. En un entorno digital cada vez más conectado y exigente, las instituciones financieras están obligadas a reforzar la protección de sus sistemas y a asegurar el cumplimiento de normativas como ISO/IEC 27034, PCI DSS o estándares locales de protección de datos.

Los costos económicos asociados a las brechas de seguridad en banca abarcan desde la inversión en equipos de respuesta a incidentes hasta la indemnización a clientes afectados y la eventual interrupción de servicios críticos (Salam et al., 2021). Asimismo, las consecuencias reputacionales pueden ser aún más difíciles de subsanar, ya que un evento de exfiltración de información o un ataque de denegación de servicio repercute directamente en la imagen de la entidad y en la credibilidad de sus plataformas tecnológicas. Por otra parte, la adopción de microservicios y entornos de contenedores —si bien aporta agilidad al negocio— incrementa la superficie de ataque, pues cada servicio puede convertirse en un punto de entrada para el atacante (Zhang et al., 2022). Esto acentúa la importancia de integrar la seguridad desde la fase inicial de desarrollo y de monitorear continuamente la integridad de las aplicaciones, tal como promueven las prácticas de DevSecOps.

2.2.2 Fundamentos de DevSecOps

El paradigma DevSecOps surge como una evolución del enfoque DevOps, integrando prácticas de seguridad informática desde las fases iniciales del desarrollo hasta la operación y el mantenimiento de las aplicaciones (Rahman & Williams, 2019). Mientras DevOps se centra principalmente en la colaboración entre desarrollo y operaciones para agilizar la entrega de software, DevSecOps añade una tercera dimensión crítica: la seguridad. De esta manera, la protección de los datos y la prevención de vulnerabilidades dejan de ser procesos "aislados" o tardíos y pasan a ser tareas continuas y automatizadas en cada paso del pipeline de desarrollo.

En la práctica, la filosofía DevSecOps se basa en diversos pilares: integración temprana de la seguridad, (Mujawar & Patel, 2021) automatización de controles (Biehl et al., 2021), cultura de colaboración y responsabilidad compartida (Rahman & Williams, 2019), monitoreo y observabilidad continua (Ventura et al., 2021),

Al introducir procesos de seguridad como código, DevSecOps fomenta la consistencia en la aplicación de políticas y lineamientos de protección. Las reglas de firewall, por ejemplo, pueden definirse como parte de la infraestructura declarativa, de modo que al realizar despliegues en entornos de microservicios, la configuración cumpla automáticamente con los estándares establecidos. Asimismo, la integración de herramientas open source para el escaneo de vulnerabilidades—como SonarQube, Trivy o OWASP ZAP—refuerza el control continuo y se adapta a la agilidad requerida por las entregas rápidas que caracterizan las arquitecturas de microservicios (Mujawar & Patel, 2021).

2.2.2.1. Principios y Prácticas Clave de DevSecOps.

El movimiento DevOps emergió a principios de la década de 2010 como una respuesta al desfase y la falta de colaboración entre los equipos de desarrollo y los de operaciones, cuyos silos generaban retrasos en la entrega de software y entorpecían la rápida adaptación al cambio (Kim et al., 2016). A través de prácticas como la Integración Continua (CI) y la Entrega Continua (CD), DevOps buscó agilizar la liberación de versiones, al tiempo que fomentaba la comunicación fluida y la responsabilidad compartida sobre la calidad del producto. Sin embargo, en sus inicios, la seguridad se

mantenía como un proceso aislado, en ocasiones solo abordado una vez que el software se encontraba en fases finales de despliegue o incluso en producción.

Este vacío motivó la aparición del concepto DevSecOps, el cual aplica el principio de "Shift-left security": incorporar pruebas y controles de seguridad desde la fase de codificación, para detectar vulnerabilidades con antelación y reducir los costos asociados a su remediación tardía (Rahman & Williams, 2019). En lugar de ver la seguridad como un obstáculo, DevSecOps promueve su integración continua en el pipeline de desarrollo, a través de herramientas de análisis estático y dinámico (SAST y DAST), escaneo de dependencias y contenedores, y validaciones automáticas en cada commit.

En la práctica, esta evolución implica un enfoque cultural que trasciende las meras herramientas tecnológicas y busca romper silos no solo entre desarrollo y operaciones, sino también entre estos y los equipos de seguridad. Se fomenta la colaboración temprana, la retroalimentación ágil y la responsabilidad compartida sobre la protección de la información en todas las fases del ciclo de vida (Biehl et al., 2021). De este modo, DevSecOps se erige como una extensión natural de DevOps, transformando la seguridad en un elemento integral de los procesos de integración y entrega continua, al tiempo que provee mecanismos de monitorización y respuesta a incidentes para salvaguardar la confiabilidad de los servicios en entornos cada vez más distribuidos y dinámicos.

2.2.2.2. Retos y Controversias.

A pesar de los múltiples beneficios que ofrece DevSecOps, su implementación no está exenta de desafíos ni controversias. Uno de los principales retos es la resistencia cultural y organizacional, dado que el enfoque implica una profunda transformación en la forma en que se concibe y gestiona la seguridad (Rahman & Williams, 2019). Equipos de desarrollo, operaciones y seguridad deben trabajar de manera colaborativa y continua, abandonando los silos y adoptando prácticas integradas que a menudo chocan con estructuras jerárquicas o procesos rígidos preexistentes. Este cambio cultural exige un liderazgo decidido, la capacitación de los profesionales y, en muchos casos, la redefinición de roles y responsabilidades.

Otro punto de controversia es la dificultad de medir el Retorno de la Inversión (ROI) en seguridad. Mientras que los costos de herramientas y recursos en DevSecOps son

relativamente tangibles, los beneficios —como la mitigación de riesgos o la protección de la reputación corporativa— son menos evidentes en términos monetarios. Estudios recientes apuntan a la necesidad de establecer indicadores de eficiencia (por ejemplo, reducción en los tiempos de corrección de vulnerabilidades, descenso de incidentes críticos en producción) que permitan justificar la inversión y evidenciar el impacto positivo en la organización (Biehl et al., 2021). Asimismo, se observa una falta de lineamientos universales para la adopción de DevSecOps, lo que dificulta la estandarización de buenas prácticas, metodologías y herramientas (Shahin & Babar, 2021). Si bien existen guías y frameworks (p. ej., OWASP SAMM, NIST), no se dispone de un consenso globalizado que especifique cómo integrar los aspectos culturales, técnicos y regulatorios en diferentes entornos empresariales. Esto conduce a implementaciones diversas que, aunque comparten principios comunes, pueden variar significativamente en cuanto a profundidad, automatización y alcance.

2.2.3 Arquitectura de Microservicios y Contenedores

La arquitectura de microservicios surge como una alternativa a los sistemas monolíticos tradicionales, buscando mejorar la escalabilidad, la flexibilidad y la mantenibilidad de las aplicaciones (Dragoni et al., 2017). En este enfoque, la aplicación se descompone en servicios independientes, cada uno encargado de una función o dominio específico. Esta separación permite desarrollar, desplegar y actualizar cada componente de manera autónoma, acelerando los ciclos de entrega y la respuesta a los cambios del mercado. Sin embargo, el mismo desacoplamiento que favorece la agilidad introduce nuevos retos, puesto que cada microservicio se convierte en un potencial punto de falla o vector de ataque si no se establecen controles de seguridad adecuados.

Uno de los catalizadores de la adopción de microservicios ha sido el auge de los contenedores, que proporcionan entornos ligeros y portables para empaquetar y ejecutar aplicaciones. Soluciones como Docker permiten abstraer las dependencias, la configuración y los archivos necesarios para un servicio determinado, facilitando la migración entre entornos de desarrollo, prueba y producción (Ma et al., 2020). Al combinar contenedores con plataformas de orquestación (por ejemplo, Kubernetes), las organizaciones pueden escalar los servicios de forma dinámica, equilibrar la carga de trabajo y aislar fallos en servicios concretos. No obstante, la granularidad y la rapidez con

que se despliegan nuevos contenedores también incrementan la superficie de ataque, sobre todo si se omiten prácticas como la ejecución de contenedores con privilegios mínimos o la protección de datos sensibles a través de secretos gestionados de manera centralizada.

En el plano de la seguridad, la arquitectura de microservicios y contenedores introduce desafíos particulares como la comunicación distribuida, gestión de configuraciones y secretos, actualización y parcheo continuo y control de acceso y segmentación de redes. En este panorama, DevSecOps ofrece un marco idóneo para la automatización de pruebas en cada etapa del pipeline (SAST, DAST, escaneo de dependencias y contenedores) y el monitoreo continuo de la infraestructura. Además, al fomentar la cultura de colaboración, se facilita la identificación temprana de riesgos y la introducción de buenas prácticas en la construcción y operación de servicios. Por lo tanto, la adopción de microservicios y contenedores puede potenciar la velocidad de entrega y la resiliencia del sistema, siempre que se establezcan lineamientos claros de seguridad en cada paso, desde el diseño hasta el despliegue en producción.

2.2.3.1. Definición y Características de Microservicios.

La arquitectura de microservicios consiste en la descomposición de una aplicación en múltiples servicios pequeños e independientes, cada uno centrado en una funcionalidad específica y que se comunica con los demás a través de protocolos ligeros, típicamente HTTP o mensajería asíncrona (Dragoni et al., 2017). Este enfoque contrasta con el modelo monolítico, en el que toda la lógica de la aplicación se despliega como una sola unidad, compartiendo recursos y ciclos de desarrollo. En un monolito, pequeños cambios en una parte del sistema pueden requerir la recompilación y el redeployment de toda la aplicación, mientras que en microservicios es posible actualizar o reescalar únicamente los componentes necesarios.

2.2.3.1.1 Beneficios.

Escalabilidad: Cada microservicio puede escalarse de forma independiente, lo que permite adaptar los recursos a la demanda específica de cada módulo sin sobredimensionar la aplicación completa (Balalaie et al., 2016).

Independencia de Despliegue: Al estar desacoplados, los microservicios facilitan ciclos de entrega continua (CI/CD). Un servicio puede lanzarse o revertirse sin interrumpir el funcionamiento de los demás, acelerando la respuesta ante errores o nuevas necesidades.

Flexibilidad Tecnológica: Los equipos pueden elegir diferentes lenguajes, frameworks y bases de datos según las exigencias de cada servicio, evitando la dependencia de un stack único.

2.2.3.2. Retos.

Comunicación entre Servicios: El aumento de llamadas de red conlleva latencias y posibles fallos de conectividad. Un error en un servicio puede propagarse rápidamente al resto si no se implementan mecanismos de resiliencia y patrones como el Circuit Breaker (Newman, 2021).

Autenticación Distribuida: En una arquitectura monolítica, la autenticación suele ser un único proceso centralizado. Con microservicios, resulta imprescindible contar con soluciones de identidad y acceso (p. ej., OAuth, JWT) que permitan la verificación de credenciales en cada servicio (Zhang et al., 2022).

Trazabilidad de Datos: El seguimiento de las transacciones, la detección de problemas de rendimiento y la auditoría de logs se complican al existir múltiples puntos de entrada y almacenamiento. Herramientas de monitoreo distribuido se vuelven esenciales para mantener la visibilidad y resolver incidentes.

2.2.3.3. Contenedores y Orquestación.

La contenedorización se ha convertido en una práctica esencial para el despliegue de aplicaciones bajo la arquitectura de microservicios. Herramientas como Docker permiten aislar un proceso junto con sus dependencias (librerías, archivos de configuración, binarios) en un entorno reproducible y portátil, conocido como contenedor (Merkel, 2014). A diferencia de las máquinas virtuales, los contenedores comparten el mismo núcleo del sistema operativo, lo que reduce el uso de recursos y facilita la migración entre distintos entornos (desarrollo, pruebas y producción). Sin embargo, esta ligereza implica

riesgos si no se aplican controles de seguridad adecuados: por ejemplo, la ejecución de contenedores con privilegios excesivos o el uso de imágenes base desactualizadas puede exponer el host a vulnerabilidades (Ma et al., 2020).

Para maximizar los beneficios de la contenedorización, las organizaciones adoptan plataformas de orquestación, siendo Kubernetes una de las más populares. A través de un plano de control (Control Plane) y múltiples nodos, Kubernetes automatiza tareas como la distribución de cargas, la creación y el escalado de contenedores y la recuperación ante fallos (Burns et al., 2016). Además de Kubernetes, existen alternativas como Amazon Elastic Container Service (ECS) o OpenShift, que proporcionan sus propios mecanismos de despliegue y administración. Todas estas soluciones comparten ventajas como la escalabilidad automática (auto-scaling) y la resiliencia ante fallas de los nodos. Sin embargo, el uso de orquestadores también conlleva problemas de configuración, especialmente en cuanto a:

Control de acceso basado en roles (RBAC): Una mala definición de permisos puede permitir que usuarios o servicios no autorizados accedan o modifiquen recursos críticos.

Gestión de secretos: Contraseñas, llaves o certificados deben almacenarse de forma segura y distribuirse únicamente a los contenedores que los requieran, evitando su exposición en archivos de configuración o variables de entorno sin protección (Zhang et al., 2022).

Redes internas: Con microservicios repartidos en múltiples contenedores, la comunicación interna cobra gran relevancia. Configurar adecuadamente los Network Policies y los servicios de malla de red (Service Mesh) resulta esencial para restringir el tráfico a lo estrictamente necesario y aislar potenciales brechas.

2.2.4 Seguridad Distribuida en Microservicios

La adopción de arquitecturas de microservicios incrementa la superficie de ataque al distribuir la lógica de negocio en múltiples servicios que se comunican entre sí, de manera que cada componente requiere autenticación y autorización independientes. En este contexto, mecanismos como OAuth 2.0 y JSON Web Tokens (JWT) facilitan el control de acceso, ya que permiten a los servicios validar tokens con información sobre la identidad y privilegios del usuario sin necesidad de mantener sesiones centralizadas

(Zhang et al., 2022). Por otro lado, la implementación de mTLS (mutual TLS) añade una capa de cifrado y autenticación mutua entre los servicios, previniendo ataques de intermediarios y asegurando que las peticiones provengan de entidades confiables (Ma et al., 2020).

Para reforzar la seguridad en infraestructuras distribuidas, se promueve el enfoque de Zero Trust, que asume la inexistencia de una "zona confiable" dentro de la red. De acuerdo con este principio, cada servicio o usuario debe ser validado explícitamente antes de permitir la comunicación, sin importar su ubicación en la topología (Cheng et al., 2021). La segmentación de redes es otro factor fundamental, pues restringe el movimiento lateral de un atacante: en lugar de mantener una gran red plana, se definen perímetros lógicos o micro perímetros que exigen políticas específicas de acceso. Este diseño granular reduce la exposición de servicios internos y limita el impacto de una posible brecha en un microservicio.

La automatización de parches y la actualización continua de los servicios son también críticas para reducir la ventana de tiempo en que las vulnerabilidades pueden ser explotadas. Como parte de la cultura DevSecOps, se recurre al escaneo periódico de dependencias y contenedores para detectar versiones inseguras o bibliotecas obsoletas, y se ejecutan pipelines de despliegue automatizados que aplican las correcciones (Rose et al., 2020). Al combinar estas prácticas con un monitoreo y una observabilidad continua —por medio de herramientas que recogen logs, métricas y trazas—, las organizaciones pueden responder de forma temprana ante anomalías o intentos de ataque, robusteciendo la confianza y resiliencia de sus sistemas de microservicios.

2.2.5 Sector Bancario: Normativas y Requisitos de Cumplimiento

El sector bancario opera con información altamente sensible —datos de cuentas, historiales de transacciones, información crediticia—, lo que implica una responsabilidad reforzada para proteger la confidencialidad, integridad y disponibilidad de los activos digitales (Ahmad et al., 2015). Además de los riesgos asociados a las brechas de seguridad y la pérdida de confianza por parte de los clientes, las instituciones financieras deben atender un entorno regulatorio riguroso que exige la adopción de procedimientos y controles específicos.

Entre las normativas y estándares internacionales más relevantes destaca el Payment Card Industry Data Security Standard (PCI DSS), orientado a proteger la información de tarjetas de pago y a estandarizar las medidas de seguridad en sistemas que procesan datos de tarjetas (Ventura et al., 2021). Asimismo, la ISO/IEC 27034 define directrices para la seguridad de las aplicaciones a lo largo de su ciclo de vida, contemplando la gestión de riesgos y la integración de prácticas de protección en cada fase de desarrollo (ISO, 2018). Estos lineamientos se combinan, en muchos países, con disposiciones locales —por ejemplo, leyes de protección de datos personales o normativas emitidas por las superintendencias bancarias— que demandan auditorías periódicas y la presentación de evidencias de cumplimiento.

La razón fundamental de estos requisitos más estrictos radica en la alta criticidad del servicio bancario y el impacto que las fallas de seguridad pueden tener, tanto en la estabilidad económica como en la credibilidad de las instituciones. Una brecha de datos puede desencadenar sanciones financieras, procesos judiciales y daños reputacionales difíciles de revertir (Ahmad et al., 2015). Por este motivo, la adopción de DevSecOps en el sector bancario no solo es un paso lógico para agilizar la entrega de software y minimizar vulnerabilidades, sino también una estrategia para mantener un nivel consistente de conformidad normativa. El pipeline de CI/CD, combinado con mecanismos de monitoreo continuo y validaciones automáticas, permite que las organizaciones produzcan y gestionen evidencias de seguridad —como reportes de vulnerabilidades corregidas o resultados de pruebas de penetración— que son esenciales para superar auditorías y supervisiones.

En entornos microservicios, la importancia de estos controles se ve magnificada: cada servicio puede tener exigencias de cifrado, autenticación y almacenamiento de datos distintas, y un error de configuración puede exponer información crítica. Bajo estas condiciones, herramientas open source y metodologías integradas de seguridad aportan la transparencia y adaptabilidad necesarias para cumplir con los requisitos locales e internacionales, al tiempo que se mantiene la agilidad requerida en el negocio bancario (Ventura et al., 2021).

2.2.6 Regulaciones Específicas en la Banca

El sector bancario está sujeto a estrictos requisitos de cumplimiento diseñados para salvaguardar los datos y las transacciones de los clientes. Una de las regulaciones más reconocidas a nivel internacional es el Payment Card Industry Data Security Standard (PCI DSS), orientado específicamente a la protección de la información de tarjetas de crédito y débito (Ventura et al., 2021). Dicho estándar establece requisitos técnicos y organizativos —por ejemplo, cifrado de datos en tránsito y en reposo, segmentación de redes y monitoreo continuo— para garantizar la integridad de los procesos de pago. Las entidades financieras que aceptan procesan o transmiten datos de tarjetas deben demostrar conformidad con PCI DSS para evitar sanciones por parte de los esquemas de tarjetas y, sobre todo, para mantener la confianza de los usuarios.

Además de los marcos globales, las normativas locales suelen imponer controles adicionales. En varios países existen entidades reguladoras como la Superintendencia de Bancos, encargadas de supervisar la solidez financiera y la seguridad de las instituciones que operan en sus respectivos territorios (Mohan & Yapa, 2019). Algunas regiones también cuentan con leyes de protección de datos que exigen la obtención de consentimientos explícitos, la notificación de incidentes de seguridad en plazos cortos y la implementación de salvaguardias específicas para datos personales. El incumplimiento de estos lineamientos puede conllevar multas significativas y la pérdida de licencias para operar.

Un aspecto clave de estas regulaciones es la exigencia de alta disponibilidad y confiabilidad en los servicios bancarios. Los organismos supervisores demandan planes de continuidad de negocio, políticas de recuperación ante desastres y auditorías periódicas para asegurar que la infraestructura sea capaz de mantener la operación en caso de fallas o ataques (Ahmad et al., 2015). En un contexto de microservicios, este requisito subraya la importancia de garantizar la seguridad no solo en el desarrollo del software, sino también en la orquestación y en la interconexión de los diferentes servicios. El enfoque DevSecOps, al automatizar pruebas de seguridad y fomentar la colaboración entre desarrollo, operaciones y equipos de cumplimiento, puede contribuir de manera decisiva a satisfacer estos elevados estándares de integridad y disponibilidad en la banca.

2.2.6.1. Estándares Internacionales.

Para complementar las regulaciones específicas del sector bancario, estándares internacionales reconocidos proporcionan directrices y buenas prácticas orientadas a la seguridad en el ciclo de vida del software. Entre los más destacados se encuentra la norma ISO/IEC 27034, la cual define un marco para la aplicación de controles de seguridad adaptados a las características de cada organización y proyecto (ISO, 2018). Este estándar va más allá de la simple evaluación de riesgos e incluye la noción de "Application Security Controls" (ASC) que deben considerarse en cada etapa, desde el análisis de requisitos hasta la liberación y operación del software. Para el entorno bancario, ISO/IEC 27034 establece una base sólida para documentar y evaluar la efectividad de las prácticas DevSecOps, demostrando cumplimiento ante auditorías e inspecciones regulatorias.

Por su parte, la comunidad OWASP (Open Web Application Security Project) ofrece un conjunto de recursos y metodologías que complementan los marcos formales. Destacan el Software Assurance Maturity Model (SAMM) y el Application Security Verification Standard (ASVS).

OWASP SAMM sirve como una guía de mejora continua, estableciendo niveles de madurez en la práctica de desarrollo seguro. Su enfoque abarca desde la gobernanza y la definición de políticas hasta la validación de la seguridad y la gestión de incidentes (OWASP Foundation, 2021).

OWASP ASVS, por su parte, proporciona un conjunto de requisitos y controles específicos para evaluar la solidez de la seguridad en aplicaciones web, categorizados por niveles de rigor. Estas herramientas facilitan a las instituciones bancarias —que manejan aplicaciones web y APIs con datos financieros— la adaptación de controles según su criticidad y el registro de evidencias en el pipeline de CI/CD. Además de ISO/IEC 27034 y OWASP, otros marcos como las guías del NIST SP 800 (elaboradas por el Instituto Nacional de Estándares y Tecnología de Estados Unidos) y los CIS Controls (Center for Internet Security) pueden resultar pertinentes en entornos bancarios. NIST SP 800-53, por ejemplo, detalla controles técnicos y organizativos para sistemas de información, mientras que los CIS Controls priorizan acciones concretas para mitigar amenazas frecuentes (NIST, 2020; CIS, 2021). Si bien cada marco aborda la seguridad desde distintas perspectivas, su aplicación combinada ofrece a las instituciones financieras un

enfoque integral para asegurar la infraestructura y los procesos de desarrollo, evidenciando el cumplimiento frente a entes reguladores y auditores internos.

2.2.6.2. Desafíos de Implementación.

La adopción de DevSecOps en instituciones bancarias conlleva un delicado balance entre la agilidad propia de los despliegues continuos y el cumplimiento estricto de los requerimientos regulatorios (Biehl et al., 2021). Por un lado, la filosofía DevSecOps promueve la integración temprana de la seguridad, la automatización de pruebas y la entrega frecuente de software, lo cual resulta clave para responder con rapidez a la evolución del mercado financiero y a las crecientes amenazas cibernéticas. Por otro lado, los organismos supervisores y las normativas locales e internacionales exigen la aplicación de controles formales y la validación exhaustiva de procesos, lo que puede prolongar los ciclos de revisión y aprobación.

Uno de los retos más marcados es la necesidad de garantizar la trazabilidad de los cambios y la generación de evidencias concretas de cada fase del ciclo de vida de desarrollo. Las entidades regulatorias suelen requerir auditorías periódicas y la demostración de que las vulnerabilidades detectadas se han corregido oportunamente, así como la prueba de que todos los accesos y despliegues se han realizado conforme a las políticas internas de seguridad (Ventura et al., 2021). Para afrontar este desafío, las prácticas DevSecOps deben complementarse con herramientas y procedimientos que registren de manera detallada cada commit, prueba de seguridad, aprobación y despliegue. De este modo, las organizaciones pueden responder con agilidad a los cambios del entorno y, al mismo tiempo, cumplir los lineamientos de supervisión, evitando sanciones y daños reputacionales asociados al incumplimiento normativo.

2.2.7 Herramientas Clave en el Entorno DevSecOps

2.2.7.1. Planificación y Gestión de Código.

2.2.7.1.1 GitHub Dashboard: Issues, Proyectos y Organización de Tareas.

En un entorno DevSecOps, la coordinación ágil y transparente de las actividades del equipo es fundamental para integrar la seguridad desde las fases iniciales del

desarrollo. GitHub Dashboard proporciona un conjunto de herramientas que facilitan la planificación y el seguimiento de tareas a través de issues, proyectos y milestones (Kalliamvakou et al., 2014). Mediante los issues, los desarrolladores y el personal de seguridad pueden describir bugs, vulnerabilidades o mejoras, etiquetándolos por prioridad o tipo de incidencia. A su vez, la funcionalidad de Proyectos (Projects) permite organizar las tareas en tableros kanban y asignar responsables, ofreciendo una vista general del progreso y la carga de trabajo. Esta trazabilidad simplifica la generación de evidencias sobre acciones correctivas y la vinculación de pruebas de seguridad con incidentes concretos, un aspecto clave para cumplir con requerimientos normativos en el ámbito bancario.

2.2.7.1.2 GitHub (Versionamiento de Código)

El versionamiento de código con Git constituye la base para la colaboración efectiva en proyectos que involucran múltiples equipos —desarrollo, operaciones y seguridad—, característica esencial en el ciclo de vida DevSecOps (Pham et al., 2020). GitHub aprovecha la arquitectura distribuida de Git, lo que permite a cada colaborador mantener una copia completa del repositorio y trabajar de forma autónoma, reduciendo los conflictos y posibilitando la recuperación en caso de fallos. Algunas ventajas clave incluyen:

Branching Models: Prácticas como Gitflow o GitHub Flow facilitan la separación de tareas, la integración continua y el control de versiones. Los equipos de seguridad pueden crear ramas específicas para la resolución de vulnerabilidades o la ejecución de pruebas (sast, dast), manteniendo el código principal estable.

Pull Requests (PRs): Este mecanismo promueve la revisión de código antes de fusionarlo a la rama principal (main). Así, los equipos de desarrollo y seguridad pueden discutir cambios, realizar comentarios y solicitar correcciones, garantizando que los parches o mejoras cumplan con los estándares de calidad y protección (Castelluccio et al., 2019). Además, la integración con sistemas de automatización —por ejemplo, GitHub Actions— permite desencadenar pipelines de seguridad en cada PR, reduciendo el tiempo de detección de vulnerabilidades.

2.2.7.2. Construcción y Testing de Código.

El versionamiento de código con Git constituye la base para la colaboración efectiva en proyectos que involucran múltiples equipos —desarrollo, operaciones y seguridad—, característica esencial en el ciclo de vida DevSecOps (Pham et al., 2020). GitHub aprovecha la arquitectura distribuida de Git, lo que permite a cada colaborador mantener una copia completa del repositorio y trabajar de forma autónoma, reduciendo los conflictos y posibilitando la recuperación en caso de fallos. Algunas ventajas clave incluyen:

Branching Models: Prácticas como Gitflow o GitHub Flow facilitan la separación de tareas, la integración continua y el control de versiones. Los equipos de seguridad pueden crear ramas específicas para la resolución de vulnerabilidades o la ejecución de pruebas (sast, dast), manteniendo el código principal estable.

Pull Requests (PRs): Este mecanismo promueve la revisión de código antes de fusionarlo a la rama principal (main). Así, los equipos de desarrollo y seguridad pueden discutir cambios, realizar comentarios y solicitar correcciones, garantizando que los parches o mejoras cumplan con los estándares de calidad y protección (Castelluccio et al., 2019). Además, la integración con sistemas de automatización —por ejemplo, GitHub Actions— permite desencadenar pipelines de seguridad en cada PR, reduciendo el tiempo de detección de vulnerabilidades.

2.2.7.3. Integración y Entrega Continua (CI/CD),

2.2.7.3.1 GitHub Actions: Flujos de Trabajo Automatizados e Integración de Pruebas de Seguridad

En un enfoque DevSecOps, la automatización juega un rol esencial para detectar vulnerabilidades de manera temprana y sistemática. GitHub Actions ofrece la posibilidad de crear flujos de trabajo (workflows) que se desencadenan ante eventos concretos, como la apertura de un pull request o la realización de un commit (Biehl et al., 2021). Cada workflow puede incluir etapas para compilar el código, ejecutar pruebas unitarias o de integración y, de ser necesario, realizar análisis de seguridad estático (SAST) o dinámico (DAST). Herramientas como

OWASP ZAP, Trivy o SonarQube pueden integrarse directamente en estos flujos, deteniendo el proceso de despliegue si se detectan vulnerabilidades críticas.

La flexibilidad de GitHub Actions permite definir diferentes pipelines según el tipo de rama (por ejemplo, desarrollo, pruebas, producción), habilitando la verificación de parámetros de calidad y seguridad adaptados al estado del código. Además, la integración con Secret Management (por ejemplo, GitHub Secrets) facilita la protección de credenciales o tokens de acceso, evitando su exposición en archivos de configuración o scripts de ejecución.

2.2.7.3.2 Posibles Alternativas (Jenkins, GitLab CI).

Aunque GitHub Actions ha ganado popularidad, existen otras soluciones de CI/CD ampliamente utilizadas:

Jenkins: Herramienta de automatización tradicional, de código abierto y con una gran comunidad de usuarios. Ofrece un sistema de plugins para integrar análisis de seguridad, pruebas de contenedores, etc. (Erich et al., 2017).

GitLab CI: Incluye un pipeline de CI/CD integrado en la misma plataforma de versionamiento. Facilita la configuración basada en YAML y la ejecución de runners para despliegues en distintos entornos.

2.2.8 Operaciones y Despliegue

2.2.8.1. Docker: Contenedores, Imágenes y Registros.

Para llevar a producción las aplicaciones construidas en un pipeline DevSecOps, la contenedorización se ha convertido en uno de los enfoques predominantes. Docker permite empaquetar la aplicación junto con sus dependencias y librerías en un contenedor ligero y portable (Ma et al., 2020). A través de un archivo Dockerfile, se define la configuración del sistema (librerías, puertos, variables de entorno) y se genera una imagen que puede distribuirse o almacenarse en un registro como DockerHub o en un repositorio self-hosted privado.

Imágenes Versionadas: Al igual que en el control de código, cada versión de la imagen tiene un identificador único (tag), lo que facilita revertir cambios en caso de detectar fallas o vulnerabilidades.

Seguridad en la Construcción: Es aconsejable usar imágenes base oficiales y mantener las dependencias actualizadas, implementando escaneos de vulnerabilidades en el pipeline CI/CD para detectar librerías obsoletas o configuraciones inseguras.

Registros Privados: Para proyectos bancarios o confidenciales, se prefiere alojar un registro privado con acceso controlado y políticas de retención de imágenes, garantizando un mayor control sobre la distribución de contenedores.

2.2.9 Monitoreo y Observabilidad

Grafana: Paneles de Visualización e Integración con Prometheus u Otras Fuentes de Métricas. La observabilidad en entornos DevSecOps cobra especial importancia para detectar anomalías y reaccionar con rapidez ante posibles incidentes de seguridad o fallas en el rendimiento. Grafana se ha consolidado como una de las principales herramientas de visualización de métricas, permitiendo la creación de paneles (dashboards) que integran datos provenientes de distintas fuentes, como Prometheus, InfluxDB o Elasticsearch (Bozic et al., 2021). Esto facilita el monitoreo de indicadores críticos relacionados con la disponibilidad de servicios, el consumo de recursos (CPU, memoria) o la tasa de errores.

Alertas Tempranas: Grafana permite configurar alertas basadas en umbrales o patrones de comportamiento inusuales, enviando notificaciones a canales como Slack, correo electrónico o sistemas de gestión de incidentes. En un contexto bancario, esta funcionalidad resulta esencial para identificar intentos de acceso no autorizado o picos de latencia que podrían sugerir actividades maliciosas.

Correlación de Métricas: Al presentar datos en paneles consolidados, los equipos de desarrollo, operaciones y seguridad pueden correlacionar eventos e identificar la raíz de los problemas, ya sea una saturación de red, un aumento inusual en el tiempo de respuesta o la aparición de nuevas vulnerabilidades tras un despliegue (Suratekar & Dhage, 2022).

2.2.9.1. Logging y Alertas Tempranas.

Además de métricas cuantitativas, la captura y el análisis de logs constituyen pilares clave para la observabilidad. Herramientas como Fluentd, Logstash o Loki se integran fácilmente con Grafana para centralizar y indexar registros de aplicaciones, contenedores u orquestadores (Kubernetes), permitiendo búsquedas y correlaciones rápidas. Un sistema robusto de logging y alertas tempranas refuerza la detección proactiva de incidentes, posibilitando respuestas ágiles y reduciendo el tiempo medio para la resolución (MTTR).

2.2.10 Detección de Vulnerabilidades

2.2.10.1. Trivy: Escaneo de Contenedores, Dependencias y Repositorios.

En entornos DevSecOps, la incorporación de herramientas de escaneo de vulnerabilidades es fundamental para automatizar la búsqueda de fallos de seguridad en cada etapa del ciclo de vida del software. Trivy destaca como una solución de código abierto que analiza imágenes de contenedores (Docker, OCI), dependencias de proyectos (p. ej., Node.js, Python, Java) y hasta repositorios completos en busca de librerías o configuraciones inseguras (Aqua Security, 2023). Su facilidad de uso e integración con pipelines CI/CD lo hace particularmente atractivo:

Análisis de Contenedores: Trivy revisa capas de la imagen en busca de paquetes del sistema operativo o librerías obsoletas, reportando la severidad de las vulnerabilidades encontradas (p. ej., CVE-IDs).

Escaneo de Repositorios: Al escanear archivos de configuración y dependencias, facilita la identificación de versiones inseguras de librerías populares o configuraciones potencialmente peligrosas (Aqua Security, 2023).

Integración con GitHub Actions/Jenkins: Es posible interrumpir el pipeline si se detectan vulnerabilidades críticas, reforzando el principio de “fail fast” y asegurando que el software no llegue a producción con fallos conocidos.

Opciones Complementarias: OWASP ZAP (DAST), Clair, Anchore, Snyk, etc.

Aunque Trivy brinda una cobertura amplia, existen alternativas o herramientas complementarias para distintos tipos de análisis:

OWASP ZAP (Zed Attack Proxy): Herramienta de DAST (Dynamic Application Security Testing) que simula ataques reales a la aplicación web en ejecución, buscando vulnerabilidades como inyecciones o exposiciones de información.

Clair: Especializada en el análisis de imágenes de contenedor, mantenida por la comunidad CoreOS.

Anchore: Se centra en el cumplimiento de políticas de seguridad para contenedores y la verificación de licencias de software.

Snyk: Ofrece funciones de escaneo de dependencias y contenedores, con un enfoque en la verificación de licencias open source y la priorización de vulnerabilidades basadas en el contexto de la aplicación (Snyk, 2023).

2.2.11 Relación con las Vulnerabilidades Descritas

Las arquitecturas de microservicios y los entornos bancarios plantean un conjunto de vulnerabilidades que van desde inyecciones (SQL, NoSQL) y exposiciones de datos sensibles hasta malas configuraciones en contenedores y orquestadores. El ecosistema de herramientas revisadas en secciones anteriores permite abordar estos riesgos de manera temprana y automatizada dentro del ciclo de vida DevSecOps:

2.2.11.1. SonarQube (SAST y Métricas de Calidad).

Detección de Inyecciones: A través de análisis estático, SonarQube examina patrones en el código susceptibles a inyecciones de tipo SQL o LDAP, emitiendo alertas cuando se detectan consultas sin sanitización.

Identificación de Code Smells: Ayuda a prevenir vulnerabilidades derivadas de prácticas de programación deficientes (por ej., malas validaciones de entrada o uso inadecuado de APIs).

Control de Dependencias: Al integrarse con plugins de terceros, puede advertir sobre librerías desactualizadas o propensas a vulnerabilidades (Manuj et al., 2021).

Trivy (Escaneo de Contenedores, Dependencias, Repositorios)

Dependencias Obsoletas: Analiza librerías y paquetes en imágenes de contenedor o en archivos de configuración del proyecto, reportando CVE conocidas y la severidad de cada hallazgo (Aqua Security, 2023).

Configuraciones Inseguras: Al escanear archivos Dockerfile o variables de entorno, Trivy advierte sobre credenciales expuestas o configuraciones que podrían habilitar el escalamiento de privilegios.

2.2.11.2. OWASP ZAP (DAST).

Pruebas Dinámicas de Aplicación: Simula ataques en entornos de prueba, evaluando posibles rutas de inyección o validaciones insuficientes en APIs y servicios web.

Escenarios Complejos: Permite configurar autenticaciones, cabeceras o flujos específicos de negocio, reproduciendo interacciones reales que podrían comprometer la aplicación.

Docker y Orquestadores (Kubernetes, ECS)

Prevención de Configuraciones Inseguras: Combinando políticas de seguridad (Network Policies, RBAC) con la automatización de parches, se reducen vulnerabilidades debidas a contenedores privilegiados o puertos expuestos (Ma et al., 2020).

Escaneo Continuo: Al integrar Trivy u otras herramientas en el pipeline de CI/CD, los contenedores se verifican antes de llegar a producción, detectando configuraciones erróneas o imágenes base obsoletas.

2.2.11.3. Herramientas Complementarias (Snyk, Clair, Anchore).

Verificación de Licencias y Cumplimiento: Snyk, por ejemplo, no solo detecta vulnerabilidades en librerías, sino que también revisa licencias open source, relevante para la banca donde la compatibilidad legal es esencial (Snyk, 2023).

Escaneo Focalizado de Imágenes: Clair y Anchore refuerzan la detección de paquetes inseguros en distribuciones Linux, identificando CVE de alta criticidad en contenedores listos para desplegar

2.3 Revisión de Estudios Previos (Estado del Arte)

2.3.1 Tendencias Emergentes en DevSecOps

La adopción de DevSecOps ha crecido de manera notable en sectores con altos requerimientos de seguridad, como el bancario, el gubernamental y el de salud. Varios estudios señalan que una implementación adecuada de DevSecOps no solo reduce el tiempo de corrección de vulnerabilidades, sino que además contribuye a elevar la confianza y la colaboración entre los equipos de desarrollo, operaciones y seguridad (Rahman & Williams, 2019).

En el ámbito bancario, se reportan casos de éxito en instituciones que han integrado pipelines CI/CD con herramientas de escaneo de contenedores y análisis estático para cumplir con requisitos de PCI DSS y normativas locales (Ventura et al., 2021). Los factores determinantes para la adopción exitosa incluyen:

Cultura: La alineación de objetivos y la superación de silos departamentales, formando equipos multidisciplinarios con responsabilidad compartida en la seguridad.

Automatización: La incorporación de pruebas de seguridad (SAST, DAST) y la detección temprana de fallas en cada commit, aplicando el principio de "shift-left security" (Biehl et al., 2021).

Capacitación: El entrenamiento continuo de desarrolladores y administradores de sistemas en buenas prácticas de codificación segura, uso adecuado de herramientas y conocimiento de los principales vectores de ataque (Biehl et al., 2021).

2.3.2 Comparativas o Evaluaciones de Herramientas

Varios papers han abordado la comparación de soluciones SAST como SonarQube, Fortify y Checkmarx, resaltando ventajas e inconvenientes en términos de tipos de vulnerabilidades detectadas, facilidad de integración con pipelines CI/CD y costo total de propiedad (Islam & Akter, 2021). SonarQube destaca por su naturaleza open source y la amplitud de reglas que cubre—tanto en seguridad como en calidad y Code Smells—, además de su integración fluida con GitHub y GitLab (Manuj et al., 2021). Por su parte, Fortify y Checkmarx ofrecen funcionalidades avanzadas de reporting y priorización de

riesgos, a menudo requeridas en grandes corporaciones, aunque su adopción implica costos de licenciamiento y una curva de aprendizaje en entornos muy regulados.

En cuanto a Trivy y el ecosistema de escáneres de contenedores, estudios empíricos muestran que Trivy ofrece tiempos de análisis reducidos y una integración sencilla con plataformas CI/CD como GitHub Actions, Jenkins o GitLab CI (Aqua Security, 2023). Herramientas como Clair, Anchore y Snyk plantean enfoques complementarios; por ejemplo, Clair se especializa en escanear imágenes Docker en un registro privado, y Anchore facilita la definición de políticas de seguridad para contenedores. Experiencias de buenas prácticas documentan que la ejecución automática de escaneos en cada nueva versión de la imagen y la remediación temprana de vulnerabilidades críticas disminuyen drásticamente los incidentes en producción (Ma et al., 2020).

GitHub Actions también ha sido objeto de análisis comparativos con Jenkins y GitLab CI. Mientras Jenkins es reconocido por su extensa comunidad y catálogo de plugins, GitHub Actions destaca por su integración nativa con GitHub y la simplicidad de configuración a través de flujos de trabajo declarativos (Biehl et al., 2021). Sin embargo, varios estudios señalan malas prácticas como la exposición de tokens en scripts o la ausencia de Quality Gates, que pueden derivar en la fusión de código inseguro a la rama principal si no se establece una política de verificación estricta (Rahman & Williams, 2019).

2.3.3 Controversias y Brechas de Investigación

La adopción de DevSecOps en el sector bancario, si bien se ha demostrado valiosa para minimizar riesgos y agilizar la entrega de software, enfrenta aún diversas controversias y vacíos que limitan su implementación sistemática:

2.3.3.1. Ausencia de un Marco Estándar para Medir la Efectividad de Pipelines DevSecOps en Banca.

Si bien existen métricas genéricas para estimar la madurez de los pipelines (tiempo de integración, número de vulnerabilidades abiertas/cerradas, etc.), no se dispone de un marco uniforme y globalmente aceptado que cuantifique la eficacia de un pipeline DevSecOps en términos de cumplimiento bancario o reducción de incidentes críticos

(Shahin & Babar, 2021). Esto genera disparidades al comparar organizaciones y dificulta la definición de objetivos homogéneos dentro de la industria.

2.3.3.2. Desafíos para Garantizar el Cumplimiento Normativo en la Automatización.

La banca está fuertemente regulada por normativas como PCI DSS o leyes de protección de datos, que exigen auditorías y validaciones formales (Ventura et al., 2021). Automatizar procesos—por ejemplo, despliegues continuos—sin un monitoreo riguroso y la recolección de evidencias puede resultar en incumplimiento, generando tensiones entre la agilidad buscada por DevOps y los procesos de control tradicionales.

2.3.3.3. Falta de Mediciones Claras del ROI en la Inversión en Seguridad Continua.

Mientras que las herramientas y la cultura DevSecOps implican costos (licencias, capacitación, tiempo de configuración), los beneficios son más difusos cuando se intenta medir el retorno de inversión (ROI). Estudios señalan dificultades para cuantificar cuánto se "ahorra" en incidentes no ocurridos o en multas evitadas (Rahman & Williams, 2019). La ausencia de indicadores económicos claros retrasa la adopción en organizaciones que priorizan resultados a corto plazo.

2.3.4 Hallazgos Clave para esta Investigación

A pesar de las brechas señaladas, la revisión de literatura y experiencias prácticas ofrece evidencias que dan sustento a la aproximación que esta tesis plantea:

2.3.4.1. Respaldo en la Elección de las Herramientas.

SonarQube y soluciones open source similares presentan buenos índices de detección de vulnerabilidades y facilitan la integración en pipelines CI/CD (Manuj et al., 2021). Trivy y otras herramientas de escaneo de contenedores permiten abordar directamente los puntos débiles de una arquitectura de microservicios, evitando dependencias obsoletas o

configuraciones inseguros (Aqua Security, 2023). La integración con GitHub Actions se alinea con las buenas prácticas de "shift-left security" y la flexibilidad requerida por entornos bancarios en continua evolución (Biehl et al., 2021).

2.3.4.2. Posibles Hipótesis o Suposiciones a Contrastar.

La adopción de un pipeline DevSecOps reduce de forma significativa la ventana de exposición a vulnerabilidades, frente a metodologías convencionales de desarrollo.

A mayor automatización y cultura de seguridad en equipos de desarrollo/operaciones, menor es la incidencia de fallas críticas en producción y mayor la capacidad de reaccionar ante auditorías o incidentes.

2.3.5 Discusión Teórica y Ubicación de la Investigación

2.3.5.1. Síntesis de los Conceptos Clave.

En la realización del presente trabajo de investigación, se ha constatado la relevancia de integrar DevSecOps con microservicios en el contexto bancario, caracterizado por estrictas normativas y altos riesgos de seguridad. La adopción de metodologías ágiles y la implementación de herramientas open source —por ejemplo, Trivy para el escaneo de vulnerabilidades en contenedores y SonarQube para el análisis estático de código— aportan un nivel de automatización y colaboración que puede mejorar sustancialmente la velocidad y la confiabilidad de los despliegues (Rahman & Williams, 2019; Ma et al., 2020). Del mismo modo, los requerimientos regulatorios (p. ej., PCI DSS, ISO/IEC 27034) y la necesidad de alta disponibilidad en sistemas bancarios legitiman la pertinencia de aplicar controles y pruebas de seguridad de manera continua y sincronizada con la entrega de software (Ventura et al., 2021).

Así, el fundamento conceptual de esta propuesta se fundamenta en cuatro pilares:

2.3.5.1.1 DevSecOps: Integración continua de la seguridad en cada fase del desarrollo.

Microservicios: Arquitectura que facilita la modularidad y la escalabilidad, al tiempo que complejiza la seguridad distribuida.

Normativa Bancaria: Exigencias regulatorias que demandan trazabilidad y evidencia de cumplimiento.

Herramientas Open Source: Soluciones como Trivy o SonarQube que reducen costos y permiten una integración flexible con pipelines de CI/CD.

2.3.6 Gap de Investigación

Pese a los avances identificados, persisten retos y brechas en la literatura. En particular, sobresale la ausencia de un marco estándar que evalúe de forma integral la efectividad de un pipeline DevSecOps en entornos bancarios, contemplando tanto la agilidad del despliegue como la detección y remediación de vulnerabilidades (Shahin & Babar, 2021). Asimismo, se carece de métricas claras y de mediciones económicas que arrojen evidencia sobre el retorno de inversión (ROI) en la incorporación sistemática de pruebas de seguridad. Esta ausencia dificulta la justificación y priorización de iniciativas DevSecOps en organizaciones con restricciones presupuestarias o con una visión enfocada en resultados de corto plazo.

2.3.7 Aporte Teórico Esperado

El presente trabajo busca validar la efectividad de las herramientas open source (Trivy, SonarQube, GitHub Actions, etc.) en un entorno real de desarrollo bancario con microservicios, tomando en cuenta la necesidad de evidencia tangible para auditores y entes reguladores. Se plantea contrastar los resultados de las pruebas de seguridad con los hallazgos de otros estudios, determinando en qué medida la implementación de un pipeline DevSecOps reduce la ventana de exposición a fallas críticas y facilita el cumplimiento de los estándares de la industria (Ventura et al., 2021). Además, se espera demostrar que la incorporación de estas soluciones no obstaculiza la agilidad, sino que refuerza la colaboración y la confianza en la entrega de software seguro.

2.4 Marco Legal

El sector bancario en el Ecuador está regulado por una serie de leyes, reglamentos y disposiciones que buscan garantizar la estabilidad del sistema financiero y la protección de la información de los usuarios. En el contexto de esta investigación, se destacan las siguientes normativas:

2.4.1 Ley Orgánica de Protección de Datos Personales (LOPDP)

Promulgada en 2021, la Ley Orgánica de Protección de Datos Personales (LOPDP) establece el marco para la recolección, almacenamiento y tratamiento de datos en el país, con el objetivo de salvaguardar el derecho a la privacidad y la autodeterminación informativa de los ciudadanos (Registro Oficial N.º 459, 2021). Entre los aspectos más relevantes para el desarrollo de software bancario se incluyen:

Consentimiento y Tratamiento de Datos: Exige el consentimiento expreso de los titulares para la recopilación de sus datos y la implementación de medidas de seguridad para prevenir accesos no autorizados.

Notificación de Incidentes: Obliga a las organizaciones a notificar a la autoridad de protección de datos y a los titulares en caso de brechas de seguridad que comprometan la información personal.

Sanciones: Establece un régimen sancionatorio en caso de incumplimiento, lo cual refuerza la necesidad de alinear los procesos de desarrollo con prácticas seguras (p. ej., DevSecOps).

2.4.2 Ley de Instituciones del Sector Financiero y Regulaciones de la Superintendencia de Bancos

La Ley de Instituciones del Sector Financiero (LISF) establece los principios fundamentales que rigen la constitución, organización y control de las entidades bancarias en el Ecuador (Registro Oficial, 2008). A su vez, la Superintendencia de Bancos emite disposiciones, normas de carácter general y resoluciones que exigen a las instituciones financieras adoptar controles de seguridad adecuados para prevenir fraude, lavado de

activos y otros riesgos asociados a la actividad bancaria. Entre las pautas más destacadas se incluyen:

Controles de Acceso y Autenticación: Definición de estándares mínimos de protección, autenticación multifactorial y cifrado de datos sensibles.

Gestión de Riesgos Tecnológicos: Necesidad de identificar, evaluar y mitigar los riesgos asociados al uso de tecnologías en los procesos bancarios, incluidas las plataformas de banca en línea o aplicaciones móviles.

Requerimientos de Auditoría: Realización de auditorías periódicas (internas y externas) que verifiquen la implementación de medidas de seguridad y el cumplimiento de la normativa vigente.

2.4.3 Normativas Técnicas Complementarias

Además de la ley específica de protección de datos y la legislación bancaria, existen otras normativas y guías técnicas expedidas por instituciones gubernamentales y organismos de control que inciden en la adopción de enfoques DevSecOps y la seguridad del software financiero, tales como:

Normas del Banco Central del Ecuador: Disposiciones sobre seguridad de la información en sistemas de pago, interoperabilidad y provisión de servicios financieros digitales.

Reglamentos de la Superintendencia de Compañías (en el caso de fintech o entidades con carácter mixto): Establecen criterios de gobernanza, confidencialidad y continuidad de negocio.

2.4.4 Importancia de la Convergencia entre lo Legal y lo Tecnológico

La intersección entre el cumplimiento legal y la modernización tecnológica cobra especial relevancia en la banca ecuatoriana. La adopción de DevSecOps, que integra la seguridad en todas las fases del ciclo de vida del software, debe alinearse con la LOPDP, la LISF y las normativas de la Superintendencia de Bancos, garantizando la protección de la información financiera y personal de los clientes. Por ello, la implementación de prácticas

seguras (p. ej., escaneo de vulnerabilidades, monitoreo continuo) no solo responde a una necesidad competitiva, sino también a un imperativo legal, cuya inobservancia puede derivar en sanciones, costos reputacionales y pérdida de confianza de los usuarios.

CAPITULO III

3 MARCO METODOLÓGICO

3.1 Descripción del área de estudio / Descripción del grupo de estudio

La disciplina DevSecOps (Development, Security, and Operations) representa un enfoque estratégico que integra el desarrollo de software, la seguridad informática y las operaciones en un marco unificado. Su propósito es garantizar que la seguridad sea un componente fundamental en todas las fases del ciclo de vida del software, en lugar de tratarse como una etapa separada o posterior, como ocurre en enfoques tradicionales. Esta metodología es ampliamente adoptada en el ámbito de las tecnologías de la información y las comunicaciones (TIC), ya que permite detectar y mitigar vulnerabilidades desde las primeras etapas del desarrollo, mejorando así la resiliencia de las aplicaciones y reduciendo costos asociados a correcciones tardías (Rajapakse et al., 2021). Ya en el contexto de las arquitecturas de microservicios, que dividen las aplicaciones en componentes independientes, la necesidad de garantizar la seguridad de cada uno de estos módulos se vuelve crítica. Estas arquitecturas, diseñadas para mejorar la escalabilidad y agilidad de los sistemas, presentan desafíos significativos en términos de detección y mitigación de vulnerabilidades. En este escenario, las herramientas de código abierto han surgido como soluciones destacadas debido a su flexibilidad, accesibilidad económica y el respaldo activo de comunidades globales, lo que permite abordar de manera eficiente las complejidades inherentes al desarrollo y la seguridad en entornos DevSecOps (Shahin et a., 2019).

Este estudio se centra en el uso de prácticas y herramientas DevSecOps para detectar vulnerabilidades en los ciclos de desarrollo de software de core bancario, un ámbito donde la protección de datos y la resiliencia de los sistemas son imperativos. De esta manera, se reconoce que los desarrolladores no solo buscan mejorar la velocidad y la agilidad de sus procesos mediante la adopción de microservicios, sino también mitigar los riesgos inherentes a estas arquitecturas. Por esta razón, es imprescindible fomentar una cultura de seguridad integrada en todas las fases del desarrollo, desde la planificación hasta la operación en entornos de producción.

La detección temprana de vulnerabilidades es un aspecto crítico en el desarrollo de software, ya que abordar problemas de seguridad en las etapas iniciales no solo reduce significativamente los costos de corrección, sino que también minimiza los riesgos operativos en entornos productivos. La identificación proactiva de fallos de seguridad permite a los equipos de desarrollo actuar de manera preventiva, evitando que vulnerabilidades lleguen a producción. Es por eso que la presente investigación evaluó y adaptó herramientas de código abierto, diseñadas para integrarse de manera eficiente en los flujos de trabajo de desarrollo. Estas herramientas posibilitaron la identificación y mitigación de vulnerabilidades en tiempo real, optimizando así la seguridad en entornos DevSecOps (Skyone Solutions, 2023).

Entre las herramientas usadas para este trabajo, se destacan las pruebas automatizadas de seguridad, que permiten realizar análisis exhaustivos del código fuente y de los artefactos de compilación, proporcionando una cobertura integral a lo largo de todo el ciclo de vida del software financiero. Estas pruebas, integradas en pipelines de integración y entrega continua (CI/CD), garantizan que cualquier vulnerabilidad detectada sea gestionada inmediatamente, evitando que llegue a producción. También, se aborda la automatización y monitorización continua en entornos de producción, reconociendo que la seguridad es un proceso dinámico y continuo. Para esto, se implementan herramientas de monitoreo en tiempo real, las cuales recopilan, procesan y analizan datos operativos para detectar y responder de manera ágil a posibles anomalías o actividades sospechosas, fortaleciendo la postura de seguridad operativa. Esta investigación también profundiza en el desarrollo y validación de mejores prácticas en DevSecOps, incluyendo técnicas avanzadas de análisis estático y dinámico de código, evaluación de enfoques de seguridad en entornos de microservicios, colaboración con la comunidad técnica para intercambiar conocimientos y experiencias, fomentando un aprendizaje colectivo.

El área de estudio de este trabajo se centra en la aplicación de DevSecOps con herramientas de código abierto orientadas a la detección de vulnerabilidades en software de core bancario, un campo en constante evolución que responde a los desafíos específicos de la seguridad en microservicios. Este estudio no solo contribuye a la mejora continua de la seguridad dentro de las organizaciones bancarias, sino que también genera un impacto significativo en la industria de las TIC, aportando conocimiento aplicable a entornos similares y contribuyendo al fortalecimiento de la seguridad global en el desarrollo de software.

3.2 Enfoque y Tipo de investigación

La presente investigación contempla un proceso metódico, sistemático y estructurado que tiene como objetivo dar respuesta a una serie de interrogantes, teorías, hipótesis y suposiciones que surgen sobre el uso de DevSecOps con herramientas opensource orientadas a microservicios, para la detección de vulnerabilidades en el ciclo de desarrollo de software de core bancario. Además, en su realización, se aplican los conocimientos adquiridos durante el proceso educativo para demostrar la eficacia del plan de estudios y demostrar las habilidades de los futuros profesionales de la UTN, Universidad Técnica del Norte. También se ha utilizado un enfoque mixto, que combina métodos cualitativos y cuantitativos con el objetivo de explorar y evaluar el uso de DevSecOps con herramientas de código abierto orientadas a microservicios, para la detección de vulnerabilidades en el ciclo de desarrollo de software de core bancario. Este enfoque permite abordar de manera integral las interrogantes planteadas, vinculando las percepciones y experiencias cualitativas con los resultados cuantitativos obtenidos a través de pruebas prácticas y análisis de datos.

Este trabajo se encuentra estructurado de manera que permite evaluar la eficiencia y eficacia de las herramientas de seguridad en la detección de vulnerabilidades, combinando la recolección de datos primarios mediante encuestas y entrevistas, con la ejecución de pruebas de eficiencia técnica en un entorno controlado. Este enfoque permite generar hallazgos relevantes tanto desde una perspectiva técnica como humana, alineados con los objetivos establecidos.

3.2.1 Metodología cualitativa

Este componente se centra en la comprensión de las percepciones, experiencias y desafíos de los profesionales involucrados en el desarrollo, implementación y gestión de DevSecOps en entornos de microservicios. Esto se realiza mediante entrevistas semiestructuradas realizadas a profesionales y expertos en desarrollo de software, seguridad informática y operaciones en entornos bancarios, con el objetivo de identificar los desafíos prácticos, buenas prácticas y áreas de mejora relacionadas con la integración de herramientas de código abierto en el ciclo de desarrollo. El instrumento utilizado para este fin es una guía de preguntas abiertas para explorar temas como la cultura DevSecOps, la adopción de herramientas open source y la gestión de vulnerabilidades. Las entrevistas

permiten obtener información contextual y detallada sobre los procesos actuales, así como las expectativas y perspectivas de los profesionales en relación con las herramientas evaluadas.

3.2.2 Metodología cuantitativa

El componente cuantitativo son la realización de encuestas estructuradas, realizadas con el objetivo de recopilar datos numéricos sobre la adopción, el uso y la percepción de efectividad de las herramientas DevSecOps en entornos bancarios a través de un Cuestionario con preguntas cerradas y escalas de Likert, diseñado para medir la frecuencia de uso, los resultados percibidos y las limitaciones identificadas en la implementación de DevSecOps.

Además, se realizaron pruebas de eficiencia técnica y con el objetivo de evaluar el desempeño de herramientas de seguridad de código abierto en la detección de vulnerabilidades en un entorno de desarrollo de microservicios se implementaron y ejecutaron las siguientes herramientas seleccionadas:

- Planificación: GitHub Dashboard
- Versionamiento del código: GitHub
- Compilación de código: Apache Maven
- Testing de código: SonarQube
- CI/CD: GitHub Actions
- Operaciones: Docker
- Monitoreo: Grafana
- Detección de vulnerabilidades: Trivy

Las herramientas se analizaron en un entorno controlado para medir la tasa de detección de vulnerabilidades reales, el número de falsos positivos, el tiempo promedio de análisis. De esta manera se determinó la efectividad técnica de estas herramientas en el ciclo de desarrollo de software financiero. Estas pruebas permitieron cuantificar el impacto de las herramientas en términos de eficiencia operativa y efectividad en la detección de vulnerabilidades, proporcionando métricas clave para la evaluación.

3.2.3 Población y Muestra

La población objetivo de este estudio está conformada por profesionales vinculados al desarrollo, seguridad y operaciones dentro del ciclo de vida de software bancario en instituciones financieras privadas de la ciudad de Quito. Estos profesionales desempeñan roles clave en la adopción e implementación de herramientas DevSecOps open source en proyectos de software financiero basado en microservicios.

3.2.3.1. Población

La población incluye:

- Desarrolladores de software que trabajan directamente en la construcción y mantenimiento de microservicios.
- Especialistas en seguridad informática responsables de la detección y mitigación de vulnerabilidades.
- Personal de operaciones que gestiona el despliegue y monitoreo de aplicaciones en entornos productivos.

3.2.3.2. Muestra

La investigación empleó un muestreo no probabilístico intencional, fundamentado en la relevancia de los participantes con respecto a los objetivos del estudio y su experiencia directa en el uso de herramientas DevSecOps. Este enfoque se consideró adecuado para recopilar datos representativos de profesionales que, por su rol y conocimientos, aportaran información valiosa sobre la adopción y percepción de herramientas de código abierto orientadas a la seguridad en el desarrollo de software bancario.

3.2.3.2.1 Encuestas Estructuradas (20 participantes)

Se administraron un total de 20 encuestas dirigidas a desarrolladores de software, especialistas en seguridad informática y personal de operaciones. El propósito fue recopilar datos cuantitativos acerca de la frecuencia de uso, efectividad percibida y limitaciones de la adopción de DevSecOps en entornos bancarios y de microservicios.

3.2.3.2.2 Entrevistas Semiestructuradas (10 participantes)

Con el fin de profundizar en las experiencias, desafíos y buenas prácticas relacionadas con la implementación de DevSecOps en microservicios, se realizaron 10 entrevistas con expertos seleccionados dentro de la población objetivo. Estas entrevistas ofrecieron una perspectiva cualitativa que enriqueció los resultados, permitiendo explorar con mayor detalle las motivaciones, dificultades y oportunidades identificadas por los profesionales.

3.2.3.2.3 Caso de Estudio sobre un microservicio Core Bancario para la integración del pago de servicios con terceros.

Además de las encuestas y entrevistas, se incorporó un caso de estudio de un microservicio utilizado por el core bancario para la integración del pago de servicios con terceros, con la finalidad de analizar la adopción práctica de herramientas DevSecOps en un entorno concreto. La selección de este microservicio obedeció a criterios accesibilidad y disponibilidad, dado que la mayoría de los bancos privados del Ecuador ofrecen esta funcionalidad en su core bancario. Este caso permitió observar de primera mano la aplicación de metodologías y herramientas en un contexto real, ofreciendo evidencia adicional sobre la efectividad y los retos de la implementación de DevSecOps en arquitecturas de microservicios.

3.2.3.3. Justificación de la selección de la muestra

La selección de esta muestra responde a la necesidad de obtener información cualitativa y cuantitativa directamente relacionada con la adopción de herramientas DevSecOps en entornos financieros. El tamaño y la composición de la muestra garantizan la representatividad de los roles clave involucrados en el ciclo de vida de desarrollo del software bancario y la diversidad de perspectivas, permitiendo contrastar datos cuantitativos obtenidos a través de encuestas con las percepciones y experiencias detalladas derivadas de las entrevistas. Además, la elección del caso de estudio se justifica por su accesibilidad, disponibilidad y su relevancia en el ámbito de la infraestructura de servicios, permitiendo observar la aplicación práctica de las metodologías y contrastar los hallazgos derivados de encuestas y entrevistas en un entorno organizacional específico. Este análisis en profundidad enriquece los resultados al proveer evidencias empíricas de

la viabilidad, efectividad y retos de la implantación de DevSecOps en una arquitectura de microservicios.

3.2.4 Tipos de investigación

3.2.4.1. Investigación Exploratoria

La investigación exploratoria en este estudio tiene como objetivo principal adentrarse en el análisis de la adopción y uso de herramientas DevSecOps de código abierto en entornos de microservicios para la detección de vulnerabilidades dentro del ciclo de desarrollo de software bancario. Esta fase inicial permitió identificar los aspectos clave del campo y trazar un panorama general que contextualice las prácticas actuales y los desafíos asociados, mediante la aplicación de encuestas y entrevistas realizadas a expertos en desarrollo, seguridad y operaciones, de manera que se exploran así las percepciones y experiencias en torno a la integración de estas herramientas. Este enfoque permite descubrir nuevos problemas, oportunidades de mejora y áreas de interés, estableciendo una base sólida para investigaciones futuras. En esta fase, el análisis cualitativo se centra en comprender los retos y las necesidades específicas del entorno bancario, sirviendo como punto de partida para una evaluación más detallada.

3.2.4.2. Investigación Descriptiva

La investigación descriptiva complementa la exploración inicial al proporcionar un panorama detallado y sistemático sobre las prácticas actuales, patrones de uso y efectividad de las herramientas DevSecOps. Este enfoque se centra en caracterizar los elementos clave relacionados con el uso de herramientas de código abierto para detectar vulnerabilidades en microservicios, las percepciones y experiencias de los desarrolladores y especialistas en seguridad y los resultados técnicos obtenidos en las pruebas de eficiencia de las herramientas.

La recopilación de datos a través de encuestas y entrevistas permite cuantificar aspectos como el nivel de adopción, los resultados percibidos y los retos enfrentados al implementar DevSecOps. Adicionalmente, los datos recolectados a partir de las pruebas

de eficiencia técnica proporcionan información objetiva sobre el desempeño de las herramientas, incluyendo la tasa de detección de vulnerabilidades, la cantidad de falsos positivos y el tiempo promedio de análisis. Este enfoque meticuloso facilita establecer conexiones significativas entre los hallazgos y las necesidades del entorno bancario, contribuyendo a la generación de conocimiento empírico que respalda tanto la práctica profesional como el desarrollo teórico en seguridad del software.

3.2.4.3. Investigación Bibliográfica

La investigación bibliográfica actúa como un pilar fundamental al proporcionar la base teórica que sustenta el presente trabajo. A través de una revisión exhaustiva de la literatura académica y profesional, se exploran los conceptos y enfoques relacionados con DevSecOps, herramientas de código abierto, microservicios y seguridad en el desarrollo de software bancario. Este tipo de investigación permite identificar tendencias emergentes, mejores prácticas y áreas de controversia, situando los hallazgos de esta investigación dentro del marco global de conocimientos existentes. La revisión de estudios previos también facilita la comparación de los resultados obtenidos con investigaciones similares, permitiendo validar o refutar hipótesis sobre la efectividad de las herramientas utilizadas en el ciclo de desarrollo de software.

3.2.4.4. Investigación de Campo

Esta se centra en la recolección directa de datos mediante la aplicación de encuestas, entrevistas y pruebas técnicas realizadas en un entorno controlado. Este enfoque busca analizar cómo se utilizan herramientas DevSecOps en la práctica y evaluar su impacto en la detección de vulnerabilidades.

El proceso de investigación de campo incluye encuestas estructuradas diseñadas para capturar datos cuantitativos sobre el nivel de adopción y percepción de las herramientas de código abierto entre los profesionales de desarrollo y seguridad; entrevistas semiestructuradas realizadas a expertos del sector bancario, con el propósito de profundizar en sus experiencias y perspectivas respecto al uso de herramientas DevSecOps, así como identificar los desafíos específicos en su implementación y pruebas de eficiencia técnica ejecutadas en un entorno controlado. Este enfoque de campo permite

obtener datos empíricos actualizados sobre la efectividad de las herramientas DevSecOps y su integración en el flujo de trabajo de desarrollo de software bancario.

3.2.5 Procedimiento de investigación

El procedimiento de investigación para abordar el tema de DevSecOps con herramientas open source orientadas a microservicios para la detección de vulnerabilidades en el ciclo de desarrollo de software de core bancario se llevó a cabo de manera estructurada y sistemática. Este proceso se dividió en varias fases, que abarcan desde la planificación inicial hasta la recolección, análisis y presentación de los resultados, siguiendo un enfoque que combina métodos cualitativos y cuantitativos.

3.2.5.1. Fase 1: Integración de Herramientas de Seguridad para Microservicios.

En esta fase, se lleva a cabo la investigación exploratoria, descriptiva y bibliográfica, complementada con la investigación de campo centrada en la adopción de DevSecOps y el uso de herramientas de código abierto aplicadas a microservicios. El objetivo es automatizar la detección de vulnerabilidades en el ciclo de desarrollo de software bancario, mediante la recopilación de datos a partir de encuestas o entrevistas dirigidas a profesionales que participan en el desarrollo de software.

Para garantizar la validez y confiabilidad del instrumento de recolección de datos (encuesta), se llevó a cabo un proceso de validación por juicio de expertos. Se seleccionaron seis profesionales con amplia experiencia en seguridad informática, ingeniería de sistemas y DevSecOps, quienes evaluaron la encuesta en términos de claridad, relevancia, coherencia y comprensibilidad. La validación se realizó a través de un instrumento diseñado específicamente para este propósito (Anexo 3), en el cual los expertos calificaron cada ítem de la encuesta utilizando una escala de Likert y proporcionaron observaciones cualitativas para mejorar la precisión del cuestionario. Los resultados de la validación indicaron un alto nivel de aceptación de la encuesta, con puntajes predominantes entre 4 y 5 en los distintos criterios evaluados. Las sugerencias proporcionadas por los expertos se enfocaron en la inclusión de ejemplos en ciertas preguntas y la mejora en la redacción de algunos ítems para mayor claridad. Con base en

esta retroalimentación, se realizaron los ajustes necesarios, asegurando así que el instrumento final fuera válido y adecuado para la recolección de datos en el estudio.

Estos insumos permiten actualizar la visión de las herramientas utilizadas en la práctica, así como de los desafíos y éxitos que experimentan quienes intervienen en la creación y mantenimiento de software de core bancario. Con los resultados obtenidos en la investigación de campo, se seleccionan las herramientas de seguridad de código abierto más idóneas para la integración y detección de vulnerabilidades en microservicios. Esta estrategia estará orientada al caso de estudio, enfocado en los backend de microservicios del core bancario destinados al pago de terceros, con el fin de reforzar la seguridad y efectividad de los procesos en cuestión.

Adicional, se emplea un repositorio en el cual se alojarán las fuentes del microservicio y se construirán los pipelines necesarios para la evaluación de las herramientas de código abierto. También, se desarrollarán microservicios en Java, haciendo uso de componentes de diversas librerías para habilitar el uso de websocket y webhooks a modo de microservicios, integrados en los pipelines de automatización y equipados con herramientas de escaneo de vulnerabilidades y verificación de código seguro a lo largo del ciclo de desarrollo de software de core bancario. Ya para terminar, una vez implementados los pipelines con las herramientas seleccionadas, se efectúa el análisis estático del código fuente, con el propósito de identificar vulnerabilidades potenciales en los componentes utilizados y examinar brechas de seguridad en los patrones de programación. Dicho análisis se realizará mediante un escaneo automatizado de los microservicios que conforman el backend del core bancario, garantizando así una supervisión continua de la seguridad en las diferentes etapas del desarrollo.

3.2.5.2. Fase2: Definición de procesos para la gestión de vulnerabilidades detectadas.

En esta fase, se implementarán los procesos necesarios para la gestión de las vulnerabilidades de un core bancario, estableciendo criterios basados en la clasificación de estas vulnerabilidades (críticas, altas, medias y bajas) y en su potencial impacto en la seguridad del sistema. Conforme a este orden de prioridad, se abordarán las vulnerabilidades y se elaborará un plan de mitigación de riesgos que refleje las acciones a tomar para su remediación oportuna. Finalmente, se generará un informe que documente

las vulnerabilidades identificadas y las medidas aplicadas, consolidando la información pertinente para su posterior análisis y seguimiento.

3.2.5.3. Fase3: Análisis del nivel de cumplimiento de la aplicación de DevSecOps.

En la tercera fase, se procederá a analizar y presentar una matriz de resultados que refleje el nivel de cumplimiento en la aplicación de DevSecOps, tomando como referencia la norma ISO/IEC 27034 y sus lineamientos relativos a la seguridad durante todo el ciclo de vida del desarrollo de software bancario. Este análisis, basado en la verificación de los aspectos más relevantes de la seguridad de las aplicaciones, permitirá identificar brechas y áreas que requieren mejoras a fin de fortalecer la implementación de DevSecOps en el entorno de microservicios.

3.2.5.4. Fase4: Validación del enfoque DevSecOps.-

En esta fase, se analizarán las métricas de seguridad del caso de estudio, considerando diversas perspectivas como defectos, eficiencia, mantenibilidad, portabilidad, viabilidad y seguridad. Se elaborará un documento que exponga la revisión de código en las aplicaciones modificadas o de reciente desarrollo, y se aplicarán las soluciones necesarias para corregir o mitigar las vulnerabilidades identificadas. Asimismo, se establecerán métricas de seguridad que evalúen el impacto de la implementación de DevSecOps en la calidad y seguridad del software bancario. El desarrollo de un caso de estudio práctico permitirá validar la efectividad del enfoque DevSecOps en el contexto bancario.

3.2.5.5. Fase5: Plan de integración para el desarrollo, seguridad y operaciones.

La quinta fase se centra en el diseño y la definición de los procesos de desarrollo, seguridad y operaciones, conformando un flujo de trabajo DevSecOps en un entorno de pruebas donde se ejecuta experimentos con herramientas open source y se analiza sus resultados. Con ello, se formula un plan de integración que describa de manera clara y concisa cómo se llevarán a cabo las actividades de desarrollo, seguridad y operaciones de

forma coordinada y automatizada. Y ya finalmente se documentarán todos los hallazgos de la investigación en un informe detallado que expondrá los objetivos, la metodología aplicada, los resultados obtenidos y las conclusiones derivadas, así como las implicaciones prácticas y teóricas. Dicho informe se difundirá entre la comunidad académica y profesional para contribuir al progreso del conocimiento en el ámbito de DevSecOps y la seguridad del software en entornos de microservicios, particularmente en el backend de un core bancario.

3.2.6 Técnicas de análisis de datos

Para el análisis de los datos cuantitativos recopilados a través de las encuestas estructuradas, se emplearon técnicas de estadística descriptiva, esto incluyó la tabulación y organización de los datos, la creación de gráficos y tablas de frecuencia, y el cálculo de medidas como porcentajes, promedios y distribuciones para interpretar patrones y tendencias clave en las respuestas de los participantes. Estas técnicas permitieron identificar los niveles de adopción, percepción y uso de las herramientas DevSecOps por parte de los profesionales involucrados en el desarrollo y seguridad de software bancario facilitando el análisis y la visualización de los datos de manera clara y estructurada. Por otro lado, los datos cualitativos provenientes de las entrevistas semiestructuradas fueron analizados mediante un enfoque de análisis de contenido, este proceso incluyó la transcripción de las entrevistas, la categorización de las respuestas según temas clave, y la identificación de patrones, coincidencias y diferencias entre las perspectivas de los participantes. Este análisis permitió comprender las experiencias, desafíos y buenas prácticas en la implementación de herramientas DevSecOps en entornos de microservicios. La triangulación entre los datos cualitativos y cuantitativos fortaleció la validez de los resultados, proporcionando una visión integral sobre el impacto y la eficacia de estas herramientas en el ciclo de desarrollo de software bancario.

3.3 Consideraciones Bioéticas

En el marco de la implementación de DevSecOps con herramientas open source orientadas a microservicios para la detección de vulnerabilidades en el ciclo de desarrollo de software de core bancario en bancos privados de la ciudad de Quito, se adoptaron una

serie de principios bioéticos que guiaron tanto la investigación como la práctica profesional. Estos principios garantizaron el respeto por los derechos de los participantes y la protección de la información involucrada en el estudio.

3.3.1 Respeto a la autonomía

Se garantizó que los desarrolladores, especialistas en seguridad y otros profesionales participantes fueran plenamente informados sobre el propósito del estudio, los métodos utilizados, y los posibles riesgos y beneficios asociados con la implementación de las herramientas DevSecOps.

3.3.2 Confidencialidad y privacidad

La confidencialidad y privacidad de los datos fueron pilares esenciales en esta investigación. Para proteger la información sensible recopilada y procesada durante el estudio, se adoptaron las medidas de anonimización de datos (toda la información recolectada, ya fuera de encuestas, entrevistas o pruebas técnicas, se trató de manera que no fuera posible identificar a las personas ni a las organizaciones participantes), uso responsable de los datos (los datos se emplearon exclusivamente para los fines previstos en la investigación y se eliminaron de manera segura una vez que dejaron de ser necesarios), cumplimiento de normativas (se garantizó el cumplimiento estricto de las regulaciones locales e internacionales de privacidad, como la Ley Orgánica de Protección de Datos Personales) y transparencia (el manejo de los datos fue transparente, permitiendo a los participantes acceder a información sobre cómo se recopilaron, procesaron y almacenaron)

3.3.3 Equidad y accesibilidad

En este estudio, se garantizó la participación equitativa de los profesionales involucrados en el desarrollo, seguridad y operaciones en el ámbito de DevSecOps, asegurando que la muestra incluyera participantes con distintos niveles de experiencia y roles dentro del ciclo de desarrollo de software bancario. Se procuró que el acceso a la investigación fuera inclusivo, proporcionando condiciones equitativas para la participación en encuestas y

entrevistas, lo que permitió recoger una diversidad de perspectivas y experiencias relevantes para el análisis.

3.3.4 Responsabilidad Social

Se consideró fundamental que esta investigación contribuyera al avance del conocimiento en seguridad informática, asegurando que los hallazgos obtenidos fueran relevantes tanto para los profesionales del sector como para la comunidad académica. El estudio buscó fortalecer la cultura de seguridad en el desarrollo de software financiero, proporcionando información valiosa sobre la efectividad de herramientas DevSecOps en la detección y mitigación de vulnerabilidades.

CAPÍTULO IV

4 RESULTADOS Y DISCUSIÓN

En esta sección se analizan los resultados obtenidos en cada una de las fases planteadas en la metodología, de la siguiente manera:

4.1 Resultados de la implementación de la Fase1: Integración de herramientas de seguridad para microservicios

A continuación, se presentan los resultados derivados de los procesos descritos en la fase 1. Primero se realizó encuestas y entrevistas y luego se utilizó un repositorio en el que se almacenaron las fuentes del microservicio, y se crearon los pipelines necesarios con las herramientas de código abierto seleccionadas; se desarrollaron además microservicios en Java que integraron componentes de diversas librerías para habilitar el uso de websocket y webhooks como microservicios, incorporándolos en los pipelines de automatización para el escaneo de vulnerabilidades y la verificación de código seguro en el ciclo de desarrollo de software bancario. Una vez implementados los pipelines con las herramientas escogidas, se realizó el análisis estático del código fuente para identificar vulnerabilidades en los componentes utilizados y revisar posibles patrones de programación inseguros, todo ello mediante el escaneo automatizado de los microservicios que conforman el backend del core bancario. Los hallazgos y la discusión de estos resultados se exponen en las secciones siguientes:

4.1.1 Resultados de las encuestas

La Tabla 1 presenta un panorama general de las respuestas registradas en las primeras seis preguntas del cuestionario, las cuales abarcan temas como el sector bancario en el que se desempeñan los participantes, su rol principal en la institución, el grado de conocimiento sobre DevSecOps, la importancia atribuida a la seguridad, la frecuencia con la que aplican prácticas DevSecOps y la percepción sobre el impacto de dichas prácticas en la seguridad de los sistemas. Al tratarse de un total de 20 participantes, esta síntesis permite vislumbrar la heterogeneidad de la muestra, así como las principales tendencias y niveles de adopción del enfoque DevSecOps en el ámbito bancario.

Tabla 1

Principales tendencias y niveles de adopción del enfoque DevSecOps en el ámbito bancario.

Pregunta	Opción 1	Opción 2	Opción 3	Opción 4	Opción 5
1. Sector (1=Banca Pública, 2=Banca Privada, etc.)	3 (15 %)	10 (50 %)	4 (20 %)	2 (10 %)	1 (5 %)
2. Rol (1=Desarrollador, 2=DevOps, etc.)	6 (30 %)	4 (20 %)	5 (25 %)	2 (10 %)	3 (15 %)
3. Conocimiento DevSecOps (1=Muy Alto, 2=Alto, etc.)	2 (10 %)	3 (15 %)	7 (35 %)	5 (25 %)	3 (15 %)
4. Importancia de la seguridad (1=Extremad. necesario, 2=Muy necesario, etc.)	7 (35 %)	9 (45 %)	3 (15 %)	1 (5 %)	0 (0 %)
5. Grado de aplicación DevSecOps (1=Siempre, 2=Frecuente, etc.)	2 (10 %)	6 (30 %)	8 (40 %)	3 (15 %)	1 (5 %)
6. Impacto en la seguridad (1=Muy efectivas, 2=Efectivas, etc.)	3 (15 %)	8 (40 %)	5 (25 %)	3 (15 %)	1 (5 %)

Los datos evidencian que la Banca Privada (50 %) representa la mayor parte del sector al que pertenecen los encuestados, mientras que el rol de Desarrollador y el de Especialista en Seguridad lideran la ocupación (30 % y 25 %, respectivamente), lo que sugiere una participación significativa de perfiles técnicos involucrados en el desarrollo seguro. En relación con el conocimiento de DevSecOps, prevalece un nivel Medio o Bajo (35 % y 25 %, respectivamente), aun cuando existe un consenso amplio (80 % combinando "Extremadamente necesario" y "Muy necesario") sobre la relevancia de integrar la seguridad en el ciclo de vida del software. Respecto a la aplicación de DevSecOps, las opciones "Siempre" y "Frecuente" suman solo el 40 %, lo que confirma la adopción parcial de dichas prácticas; sin embargo, la mayoría considera su impacto en la seguridad como Efectivo o Muy Efectivo, reflejando una percepción favorable frente al potencial de las metodologías DevSecOps, a pesar de la implementación todavía limitada en los entornos encuestados.

La Tabla 2 presenta el nivel de uso y conocimiento de tres herramientas de planificación (Open Project, GitHub Dashboard y Notion) entre los 20 encuestados. Este bloque de preguntas se enfoca en la fase inicial del ciclo de desarrollo, en la que la coordinación de tareas, la asignación de recursos y la visibilidad de proyectos resultan fundamentales para la eficiencia de los equipos.

Tabla 2
Uso y Conocimiento de Herramientas de Planificación

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Open Project	2 (10 %)	4 (20 %)	5 (25 %)	5 (25 %)	4 (20 %)
GitHub Dashboard	7 (35 %)	6 (30 %)	4 (20 %)	3 (15 %)	0 (0 %)
Notion	3 (15 %)	4 (20 %)	5 (25 %)	5 (25 %)	3 (15 %)

Los resultados evidencian que GitHub Dashboard encabeza la preferencia en cuanto a "Muy Alto" y "Alto" (35 % y 30 %, respectivamente), consolidándose como la herramienta de planificación más empleada o reconocida dentro de la muestra. En contraste, Open Project y Notion presentan niveles de adopción más moderados, con un uso y conocimiento intermedios o bajos en un 50 % de los participantes (sumando las categorías "Bajo" y "Nada" en ambos casos). Estos hallazgos indican una tendencia a centralizar la planificación en la misma plataforma que provee servicios de repositorio y CI/CD, lo que sugiere que la integración con flujos de desarrollo —y, potencialmente, con enfoques DevSecOps— constituye un factor decisivo para la elección de una herramienta de planificación.

La Tabla 3 recoge los datos sobre el nivel de conocimiento y uso de las principales herramientas de control de versiones empleadas por los 20 encuestados. En el contexto de metodologías DevSecOps, la gestión confiable y transparente de versiones representa un pilar fundamental para integrar la seguridad de forma temprana y coordinada, al evitar inconsistencias en el código y facilitar la trazabilidad de cambios.

Tabla 3
Conocimiento y uso de herramientas de Versionamiento

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Git	8 (40 %)	6 (30 %)	3 (15 %)	2 (10 %)	1 (5 %)
GitHub	7 (35 %)	7 (35 %)	4 (20 %)	2 (10 %)	0 (0 %)
GitLab	2 (10 %)	4 (20 %)	6 (30 %)	5 (25 %)	3 (15 %)

Los resultados evidencian un amplio dominio de Git (40 % "Muy Alto" y 30 % "Alto"), seguido muy de cerca por GitHub, que exhibe una adopción considerable (70 % en los niveles de "Muy Alto" y "Alto"). Esta elevada popularidad se alinea con las prácticas

actuales de integración y entrega continua, en las que la vinculación entre repositorios y pipelines es clave para automatizar la detección de vulnerabilidades y las pruebas de seguridad. Por otro lado, GitLab muestra una presencia más moderada (30 % en nivel "Medio"), lo que sugiere una menor adopción o familiaridad entre los encuestados, si bien continúa siendo una alternativa utilizada por una porción significativa de los participantes. En general, se observa una preferencia clara por herramientas con gran respaldo comunitario y amplia compatibilidad con los flujos de DevSecOps.

La Tabla 4 muestra el nivel de adopción y conocimiento de tres herramientas vinculadas a la fase de compilación y construcción de proyectos dentro del ciclo de desarrollo. Dado que las prácticas DevSecOps requieren pipelines consolidados —en los que la detección de fallas se produce tan pronto como el código se integra—, resulta esencial conocer qué utilidades emplean los equipos para gestionar las dependencias y los procesos de empaquetado de los artefactos.

Tabla 4
Conocimiento y uso de herramientas de Compilación

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Gradle	2 (10 %)	5 (25 %)	6 (30 %)	5 (25 %)	2 (10 %)
Apache Maven	7 (35 %)	6 (30 %)	3 (15 %)	2 (10 %)	2 (10 %)
Jenkins	4 (20 %)	5 (25 %)	4 (20 %)	4 (20 %)	3 (15 %)

Los datos sugieren que Apache Maven se erige como la herramienta más reconocida (65 % en la suma de "Muy Alto" y "Alto"), lo que concuerda con su extendido uso para la gestión de dependencias en entornos Java. Gradle, si bien mantiene un porcentaje similar de "Medio" o "Alto", no alcanza el mismo grado de adopción de Maven, al ubicarse en una franja moderada. Por otro lado, Jenkins refleja un estatus híbrido, pues además de compilar y empaquetar, se asocia con tareas de integración continua, lo que explica su adopción más diversificada (sumando 45 % entre "Muy Alto" y "Alto"). Estos hallazgos sugieren que, en la práctica, los equipos prefieren soluciones que combinen control de dependencias y automatización de builds con la orquestación de pipelines, potenciando así el enfoque DevSecOps en la detección temprana de vulnerabilidades.

La Tabla 5 reúne información sobre el grado de familiaridad y utilización de tres herramientas de testing, con énfasis en el análisis de la calidad y la seguridad del código. En un enfoque DevSecOps, las pruebas automatizadas, tanto de interfaz como de lógica interna, permiten reforzar la detección temprana de fallas y la integridad de los servicios bancarios, otorgando mayor confianza al ciclo de desarrollo y despliegue continuo.

Tabla 5
Conocimiento y uso de herramientas de Testing

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
SonarQube	6 (30 %)	7 (35 %)	4 (20 %)	2 (10 %)	1 (5 %)
Selenium	3 (15 %)	4 (20 %)	6 (30 %)	4 (20 %)	3 (15 %)
Gremlin	1 (5 %)	2 (10 %)	4 (20 %)	8 (40 %)	5 (25 %)

Los resultados indican que SonarQube se posiciona como la herramienta de testing más reconocida (65 % en los niveles “Muy Alto” y “Alto”), lo que coincide con la importancia de su función en el análisis estático de código y la detección de vulnerabilidades o malas prácticas. Selenium, destinada al testeo de interfaces web, presenta un uso moderado, con un 35 % de adopción en “Muy Alto” + “Alto”. En contraste, Gremlin, orientada a la ingeniería del caos y pruebas de resiliencia, exhibe un menor grado de conocimiento (15 % en sumatoria de “Muy Alto” y “Alto”), reflejando que su adopción se circunscribe a casos puntuales y es menos habitual en entornos bancarios, posiblemente debido a la complejidad de simular fallas en infraestructuras críticas.

La Tabla 6 presenta la adopción y familiaridad de los encuestados con tres herramientas de Integración y Entrega Continua (CI/CD): Jenkins, GitHub Actions y Circle CI. Dadas las exigencias del entorno bancario y el enfoque DevSecOps, la orquestación de procesos de construcción, pruebas y despliegue resulta fundamental para detectar vulnerabilidades en fases tempranas y facilitar la trazabilidad de cambios.

Tabla 6
Conocimiento/uso de herramientas de CI/CD

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Jenkins	5 (25 %)	5 (25 %)	6 (30 %)	3 (15 %)	1 (5 %)
GitHub Actions	8 (40 %)	5 (25 %)	4 (20 %)	2 (10 %)	1 (5 %)

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Circle CI	2 (10 %)	3 (15 %)	5 (25 %)	5 (25 %)	5 (25 %)

El panorama refleja una consolidación de GitHub Actions (65 % en "Muy Alto" + "Alto"), principalmente gracias a su integración directa con la plataforma GitHub. Jenkins conserva un papel relevante, con un uso moderado-alto que se justifica por su extensa trayectoria y amplio ecosistema de plugins. Circle CI, por su parte, exhibe un menor nivel de adopción (25 % en sumatoria de "Muy Alto" y "Alto"), lo cual podría atribuirse a la preferencia por soluciones que se integran de forma nativa en repositorios ampliamente utilizados o a la cultura organizacional de cada institución. En conjunto, estos hallazgos sugieren que la facilidad de configuración y la compatibilidad con el ecosistema de repositorios son factores decisivos para la adopción de CI/CD en entornos DevSecOps.

La Tabla 7 expone el nivel de adopción y familiaridad de los participantes con distintas herramientas de operaciones en entornos bancarios basados en microservicios. Estas utilidades permiten la ejecución y orquestación de contenedores, aspecto crítico en la implementación de metodologías DevSecOps, donde la entrega y el mantenimiento de los servicios requieren integración continua y alta disponibilidad.

Tabla 7
Conocimiento y uso de herramientas de Operaciones

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Docker	7 (35 %)	6 (30 %)	3 (15 %)	3 (15 %)	1 (5 %)
ContainerD	1 (5 %)	3 (15 %)	5 (25 %)	7 (35 %)	4 (20 %)
Podman	2 (10 %)	3 (15 %)	5 (25 %)	5 (25 %)	5 (25 %)

Los resultados confirman que Docker mantiene la hegemonía en el segmento de operaciones, con un 65 % de "Muy Alto" o "Alto" en su uso o conocimiento. En contraste, ContainerD y Podman exhiben valores considerablemente menores, reflejando una menor difusión en la muestra (20 % o menos en niveles altos de adopción). Estos hallazgos apuntan a que, en el contexto bancario evaluado, la preferencia por Docker responde tanto a su amplia documentación y respaldo comunitario como a la necesidad de contar con herramientas consolidadas para la seguridad y el despliegue estable de aplicaciones. Con

ello, se observa un fuerte alineamiento del sector con plataformas de contenedorización populares, lo cual favorece la integración con pipelines de seguridad y la orquestación en arquitecturas de microservicios.

La Tabla 8 presenta el grado de adopción de tres herramientas de monitoreo (Kibana, Grafana y Zabbix) dentro de la muestra encuestada. En un entorno bancario con microservicios, la observabilidad resulta esencial para detectar incidentes de seguridad y problemas de rendimiento en tiempo real. Por este motivo, la eficacia de las soluciones de monitoreo y la facilidad de integración con pipelines DevSecOps son factores determinantes para la estabilidad de las aplicaciones críticas.

Tabla 8.
Conocimiento y uso de herramientas de Monitoreo

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Kibana	3 (15 %)	4 (20 %)	6 (30 %)	4 (20 %)	3 (15 %)
Grafana	6 (30 %)	5 (25 %)	5 (25 %)	3 (15 %)	1 (5 %)
Zabbix	2 (10 %)	2 (10 %)	5 (25 %)	7 (35 %)	4 (20 %)

Los datos reflejan un uso moderado de Kibana (35 % combinando “Muy Alto” y “Alto”), en tanto que Grafana se destaca con un 55 % de adopción alta, subrayando su papel preeminente en la visualización de métricas y la configuración de paneles personalizados. Por otra parte, Zabbix muestra la menor popularidad (solo 20 % en los niveles de “Muy Alto” y “Alto”), lo que sugiere que, si bien puede considerarse una herramienta robusta, su adopción en los flujos de trabajo de la banca encuestada no es tan frecuente. Estos hallazgos ratifican la tendencia a preferir plataformas con una interfaz intuitiva y un ecosistema de plugins amplio, que faciliten la integración con contenedores y servicios de orquestación en arquitecturas de microservicios.

Los resultados presentados en la siguiente tabla reflejan el nivel de conocimiento y uso de tres herramientas open source en el ámbito de la seguridad de microservicios.

Tabla 9
Resultados de la Encuesta sobre el Uso de Herramientas Open Source para la Seguridad de Microservicios

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
Trivy	20	22	25	5	3

Herramienta	Muy Alto	Alto	Medio	Bajo	Nada
OWASP Dependency-Check	10	15	18	10	7
Anchore	8	12	16	12	9

Los resultados de la encuesta indican que Trivy es la herramienta de seguridad de microservicios más utilizada, con un 67% de los encuestados reportando un uso considerable y solo un 8% con poco o ningún conocimiento de ella. En contraste, OWASP Dependency-Check presenta una adopción moderada, con un 43% de los participantes ubicándose entre los niveles Medio y Alto, aunque un 17% aún desconoce o no utiliza la herramienta. Anchore es la menos conocida, con solo un 36% de usuarios en niveles Medio y Alto, y un 21% que no la usa en absoluto. En general, estos datos reflejan que Trivy es la herramienta predominante, posiblemente debido a su facilidad de uso y efectividad, mientras que OWASP Dependency-Check y Anchore podrían beneficiarse de una mayor promoción y capacitación dentro de la organización.

4.1.2 Resultados de las entrevistas

La siguiente tabla presenta la caracterización detallada de los 10 entrevistados seleccionados para el estudio, describiendo su rol o puesto dentro del ámbito tecnológico, su experiencia acumulada en el área de TI, los años de implementación de prácticas DevSecOps, así como el sector bancario al que pertenecen. Esta tabla proporciona una visión general de la diversidad y el perfil profesional de la muestra, lo que resulta fundamental para interpretar los resultados cualitativos obtenidos en la investigación de campo sobre la implementación de DevSecOps en entornos de microservicios en el sector bancario

Tabla 10
Caracterización de la Muestra

Entrevistado (ID)	Rol / Puesto	Experiencia en TI	Años aplicando DevSecOps	Sector Bancario
E1	Desarrollador Senior	7 años	3 años	Banca Privada
E2	Especialista en Seguridad	5 años	2 años	Banca Pública
E3	Ingeniero DevOps	6 años	3 años	Cooperativa de Ahorro y Crédito

E4	Líder de Proyecto de TI	8 años	2 años	Banca Privada
E5	Desarrollador / Seguridad	4 años	1 año	Banca Privada
E6	Ingeniero DevOps / Arquitecto	6 años	4 años	Cooperativa de Ahorro y Crédito
E7	Especialista en Microservicios	5 años	2 años	Banca Pública
E8	Desarrollador Full Stack	3 años	1 año	Banca Privada
E9	Especialista en QA / Seguridad	4 años	2 años	Banca Privada
E10	Líder de Operaciones en TI	7 años	3 años	Banca Privada

La mayoría de los entrevistados desempeñan roles críticos en el ámbito del desarrollo y la seguridad, con una preponderancia de profesionales provenientes de la banca privada. Los datos indican que la experiencia en TI de los participantes oscila entre 3 y 8 años, mientras que la adopción de prácticas DevSecOps varía entre 1 y 4 años, lo que sugiere una tendencia emergente en la integración de estas metodologías dentro del sector financiero. La inclusión de perfiles de desarrolladores senior, especialistas en seguridad y líderes de proyectos refuerza la validez de la muestra, al garantizar que las opiniones recabadas provengan de profesionales con una experiencia sustancial en la implementación de soluciones tecnológicas avanzadas. Esta diversidad en la muestra permite abordar de manera integral los desafíos y éxitos relacionados con la adopción de DevSecOps, proporcionando un fundamento robusto para el análisis comparativo de resultados en el estudio.

La siguiente tabla recoge las respuestas obtenidas de los 10 expertos entrevistados en relación con la implementación de prácticas DevSecOps en sus respectivas organizaciones. En esta tabla se evidencian tres dimensiones fundamentales: los mayores desafíos enfrentados durante la implementación, los cambios operativos y técnicos adoptados para superar dichos desafíos, y la influencia que estas prácticas han tenido en la cultura organizacional de los equipos. Este desglose permite contextualizar de manera integral el proceso de adopción de DevSecOps, resaltando tanto los obstáculos iniciales como las estrategias implementadas para impulsar la colaboración y la transformación en las organizaciones.

Tabla 11
Implementación de DevSecOps

Entrevistado (ID)	Mayores Desafíos	Cambios Operativos/Técnicos	Influencia en la Cultura Organizacional
E1	Falta de capacitación específica en seguridad	Creación de pipelines CI/CD con validaciones SAST/DAST	Incremento de la colaboración entre desarrollo y seguridad
E2	Resistencia inicial de equipos tradicionales	Adopción de contenedores y configuración automatizada de parches	Cultura más orientada a la responsabilidad compartida en la protección de datos
E3	Dificultad para integrar herramientas open source a sistemas legados	Migración progresiva hacia microservicios; uso de Docker y Kubernetes	Mejora gradual; requerimiento de apoyo directivo para impulsar el cambio
E4	Falta de una hoja de ruta clara al inicio	Definición de un plan de despliegue continuo con GitHub Actions y SonarQube	Conciencia de la importancia de la seguridad desde etapas tempranas
E5	Dificultades en la sincronización de los equipos de desarrollo y seguridad	Configuración de Quality Gates obligatorias; integración de Trivy para contenedores	Aumento de la comunicación diaria; reducción de silos
E6	Compatibilidad con normativa interna bancaria	Estandarización de pipelines y adopción de ISO/IEC 27034 como referencia	Culturalmente, mayor apertura a la colaboración transversal
E7	Falta de métricas para medir ROI en la inversión de seguridad	Uso de dashboards (Grafana) para monitorear vulnerabilidades y tiempo de corrección	La alta gerencia percibe la seguridad como parte esencial del desarrollo
E8	Escaso soporte formativo para herramientas en español	Implementación de guías internas y workshops sobre DevSecOps	Fortalecimiento del aprendizaje colaborativo
E9	Mínimo conocimiento en Microservicios	Ajustes en arquitecturas monolíticas hacia contenedores y orquestación (Docker+K8s)	Mejor entendimiento del ciclo de vida de la seguridad en cada servicio
E10	Escalabilidad de las pruebas de seguridad en pipelines	Automatización con GitHub Actions, uso de OWASP ZAP en entornos QA	Refuerzo del sentido de propiedad y responsabilidad conjunta

Esta tabla revela también. una diversidad de desafíos, donde destacan la falta de capacitación específica (E1, E8) y la resistencia inicial de los equipos tradicionales (E2). Asimismo, se observa que la implementación de soluciones técnicas, como la creación de pipelines CI/CD con validaciones de seguridad y la adopción de contenedores, ha sido clave para mitigar dichos retos. La mayoría de los entrevistados coincide en que estos cambios han generado un notable incremento en la colaboración interdepartamental, facilitando una cultura organizacional más orientada a la responsabilidad compartida en la protección de datos (E2, E5, E7). Sin embargo, también se evidenció la necesidad de

contar con métricas claras para evaluar el retorno de inversión en seguridad (E7) y la dificultad de integrar herramientas open source a sistemas legados (E3). En conjunto, los resultados sugieren que, a pesar de los retos iniciales, la implementación de DevSecOps ha contribuido significativamente a transformar los procesos internos y a mejorar la postura de seguridad en las organizaciones entrevistadas.

La Tabla 12 presenta las respuestas obtenidas de los entrevistados en relación con las herramientas de código abierto recomendadas para la implementación de prácticas DevSecOps en entornos de microservicios. La tabla se estructura en tres columnas principales: las herramientas sugeridas, los criterios que fundamentaron la elección de cada solución, y el manejo adoptado para mitigar los falsos positivos. Este desglose permite apreciar no solo las preferencias individuales en cuanto a herramientas, sino también las consideraciones técnicas y operativas que guían la selección de estas soluciones en un contexto de seguridad integral en el desarrollo de software bancario.

Tabla 12
Herramientas Open Source

Entrevistado (ID)	Herramientas Recomendadas	Criterios de Elección	Manejo de Falsos Positivos
E1	Trivy (escaneo de contenedores), SonarQube (análisis estático), GitHub Actions (CI/CD)	Facilidad de integración y costo nulo	Creación de reglas personalizadas para ignorar hallazgos repetitivos
E2	Jenkins, OWASP Dependency-Check, Docker	Comunidad activa y amplia documentación	Validación manual de los reportes críticos
E3	GitLab CI, Clair (contendedores), Grafana	Compatibilidad con la infraestructura actual	Ajuste de umbrales y configuración de alertas
E4	GitHub Dashboard, SonarQube, OWASP ZAP	Integración nativa con repositorios y pipelines	Notificación en tiempo real a desarrolladores y revisión colaborativa
E5	Gradle + Apache Maven, Trivy, Circle CI	Velocidad de ejecución y soporte multiplataforma	Filtrado de falsos positivos a nivel de pipeline
E6	Jenkins, Podman, Kibana	Escalabilidad y cumplimiento con normativas internas	Revisión manual y priorización de vulnerabilidades
E7	Docker, SonarQube, Grafana	Convergencia con las herramientas de monitoreo y orquestadores	Retroalimentación temprana en cada pull request
E8	Git + GitHub Actions, Selenium, Trivy	Facilidad de aprendizaje por parte del equipo	Uso de listas blancas/negras en análisis estático
E9	GitHub Dashboard, OWASP ZAP, Notion	Sincronización con metodologías ágiles y enfoque colaborativo	Sesiones de revisión conjunta entre Dev y Sec
E10	Jenkins Pipeline, Anchore, Kibana, Snyk	Políticas de compliance y reporte de licencias	Coordinación entre especialistas para descartar hallazgos irrelevantes

Se puede evidenciar una marcada inclinación hacia herramientas que ofrezcan una fácil integración y un costo nulo, como es el caso de Trivy y SonarQube, lo cual se refleja en la recomendación recurrente de estas aplicaciones (E1, E5, E8). Asimismo, la relevancia de contar con soluciones que faciliten la gestión de pipelines CI/CD se destaca con la mención de GitHub Actions y Jenkins. Los criterios de elección mencionados, tales como la amplia documentación, la compatibilidad con la infraestructura existente y la escalabilidad para cumplir con normativas internas, subrayan la importancia de disponer de herramientas robustas y adaptables. Por otro lado, el manejo de falsos positivos varía entre la configuración de reglas personalizadas y la validación manual, indicando que, aunque las herramientas automatizadas son esenciales, el factor humano sigue siendo crucial para priorizar los hallazgos relevantes. Estos resultados ponen de manifiesto que la adopción de herramientas open source en DevSecOps depende tanto de las capacidades técnicas de la solución como de la experiencia y el enfoque colaborativo de los equipos de desarrollo y seguridad.

La Tabla 13 recopila las percepciones de los entrevistados acerca del impacto de la adopción de DevSecOps en entornos de microservicios, específicamente en términos de efectividad, seguridad en aplicaciones bancarias y tiempos de respuesta ante amenazas. Cada entrevistado ha evaluado tres aspectos: la efectividad de DevSecOps en la detección temprana y reducción de errores, la mejora en la seguridad y conformidad con normativas (como PCI DSS e ISO/IEC 27034), y la capacidad de responder a incidentes, ya sea mediante alertas inmediatas o una reducción sustancial en los tiempos de corrección.

Tabla 13
Impacto en la Seguridad

Entrevistado (ID)	Efectividad de DevSecOps en Microservicios	Seguridad en Aplicaciones Bancarias	Tiempos de Respuesta ante Amenazas
E1	Reducción de errores en despliegues continuos al tener pruebas automatizadas	Mejora notable en la detección temprana de dependencias críticas	Alertas casi inmediatas; correcciones en <24h
E2	Mayor resiliencia, aunque requiere madurez en cultura	Refuerzo en la validación de transacciones y cifrado	Reducción del tiempo de parches de días a horas
E3	Detección de fallas tempranas en la comunicación de microservicios	Facilita la conformidad con PCI DSS en su organización	Mejoría del tiempo medio de respuesta, pero aún se ven casos de retrasos
E4	Ha disminuido la brecha de seguridad al integrar SAST/DAST en cada commit	Alineación con ISO/IEC 27034 y reportes de auditoría	Permite identificar ataques a APIs bancarias y responder con agilidad

Entrevistado (ID)	Efectividad de DevSecOps en Microservicios	Seguridad en Aplicaciones Bancarias	Tiempos de Respuesta ante Amenazas
E5	Aún en fase inicial, se prevé una caída en el número de incidencias al automatizar pruebas	Procesos de aprobación rápida cuando se corrigen vulnerabilidades de alto impacto	Respuesta ágil a incidentes, aunque la falta de personal especializado es un freno
E6	Más efectiva en contenedores; algunos fallos al escanear monolitos residuales	Favorece la integración con sistemas heredados de forma progresiva	Detección de ataques en tiempo real con Grafana+alertas
E7	Mejor control de dependencias externas, evitando librerías obsoletas	Incrementa la confianza en la capa de servicios de core bancario	Respuesta proactiva, gracias a la retroalimentación automatizada
E8	Incremento de visibilidad en cada servicio, evitando configuraciones inseguras	Aporta valor al cumplimiento normativo interno, aunque no soluciona toda la brecha	Se logra escalar problemas a la gerencia en minutos
E9	Capacidad de contener incidentes y aislar servicios, mitigando impactos mayores	Mejora la trazabilidad de vulnerabilidades en los entornos de prueba y producción	Respuesta más coordinada, aunque depende de la madurez del equipo
E10	Elevado nivel de satisfacción; el enfoque reduce la repetición de fallas conocidas	Cumple con los lineamientos de la Superintendencia de Bancos en la parte de monitoreo continuo	Detección temprana con pipelines diarios, manejo eficiente de parches

La mayoría de los expertos percibe la implementación de DevSecOps como un factor positivo en la mejora de la seguridad de las aplicaciones bancarias. Varios entrevistados señalaron que la integración de técnicas como SAST/DAST en cada commit ha contribuido a la reducción de errores y a una detección más temprana de vulnerabilidades, lo que se traduce en tiempos de respuesta significativamente menores (por ejemplo, de días a horas o incluso en menos de 24 horas). No obstante, también se identificaron desafíos, tales como la dependencia de la madurez del equipo y la necesidad de contar con personal especializado para lograr respuestas ágiles. En conjunto, estos resultados subrayan que, si bien el enfoque DevSecOps es percibido como efectivo y valioso, su éxito depende en gran medida de la integración de buenas prácticas, de la cultura organizacional y del soporte técnico continuo para mantener la confiabilidad en la detección y remediación de vulnerabilidades.

La Tabla 14 presenta las recomendaciones ofrecidas por los 10 expertos entrevistados, centradas en tres ejes fundamentales: las buenas prácticas para la adopción de DevSecOps, las áreas clave en la selección y uso de herramientas open source, y los recursos formativos o capacitaciones que consideran esenciales para la implementación

exitosa. Este resumen cualitativo permite evidenciar las estrategias y sugerencias que los profesionales han identificado para optimizar la integración de procesos de desarrollo, seguridad y operaciones en entornos de microservicios del core bancario.

Tabla 14
Recomendaciones

Entrevistado (ID)	Buenas Prácticas para Adoptar DevSecOps	Áreas Clave para Herramientas Open Source	Capacitaciones o Recursos Esenciales
E1	Definir pipelines claros y establecer Quality Gates con SonarQube	Documentar la gestión de vulnerabilidades para asegurar uso constante de Trivy	Talleres prácticos sobre CI/CD y escaneo de contenedores
E2	Fomentar la participación de equipos en etapas tempranas de seguridad	Escoger herramientas consolidadas con alta comunidad (GitHub Actions, Jenkins, Docker)	Certificaciones orientadas a DevOps y fundamentos de seguridad (ISO/IEC 27034)
E3	Realizar evaluaciones regulares de las políticas de acceso y parches	Integrar monitoreo (Grafana) y escaneo automático en cada sprint	Formación en microservicios y contenedores; cursos en Docker y Kubernetes
E4	Incluir la seguridad en cada commit y asegurar pruebas SAST/DAST en un entorno controlado	Mantenimiento de repositorios internos de imágenes y dependencias para la banca	Charlas semanales de retroalimentación con el equipo de desarrollo y seguridad
E5	Adoptar un roadmap progresivo de microservicios hacia contenedores orquestados	Verificar la compatibilidad con normativas de banca y adaptar configuraciones de Docker	Manuales internos e inducción intensiva para nuevos colaboradores
E6	Sistematizar la clasificación de vulnerabilidades y su priorización	Escoger herramientas que generen evidencia de auditoría para la Superintendencia	Workshops en detección y mitigación de vulnerabilidades web (OWASP Top 10)
E7	Enfatizar la observabilidad: logs centralizados, dashboards, alertas	Vincular resultados de escaneo con métricas de rendimiento y disponibilidad	Formación interna en ISO/IEC 27034 y PCI DSS
E8	Mantener lineamientos de codificación segura y reglas de estilo en repositorios	Usar GitHub o GitLab con integraciones sencillas de SAST y DAST	Recursos oficiales de GitHub, Docker y Trivy para aprendizaje contínuo
E9	Aprovechar las pruebas automatizadas para validaciones de seguridad en pipelines	Dar preferencia a herramientas de fácil configuración (SonarQube, Trivy, ZAP)	Sesiones de pares entre desarrolladores y expertos en seguridad
E10	Establecer un plan de integración de DevSecOps con la directiva de TI en la banca	Revisar continuamente la eficacia de las herramientas y realizar pruebas de estrés	Diplomados o especializaciones en DevSecOps y seguridad de aplicaciones bancarias

Las recomendaciones de los expertos destacan la importancia de establecer pipelines claros y Quality Gates con herramientas como SonarQube, lo cual favorece la detección temprana de vulnerabilidades. Asimismo, la elección de herramientas open source se

fundamenta en criterios como la facilidad de integración, el respaldo de comunidades activas y la capacidad para generar evidencia de auditoría, lo que es crucial en entornos regulados. Por otra parte, la mayoría de los entrevistados enfatiza la necesidad de fortalecer la capacidad formativa a través de talleres, certificaciones y sesiones de revisión en equipo, lo que demuestra una orientación hacia el desarrollo continuo de competencias técnicas y la consolidación de una cultura colaborativa.

4.1.3 Integración de herramientas de seguridad para microservicios

4.1.3.1. Ambiente entorno local.

Para levantar este ambiente en un entorno local se necesitaron las siguientes herramientas:

1. Instalar IntelliJ IDEA es el entorno de desarrollo integrado (IDE), para el desarrollo en Java lo podemos encontrar en: https://www.jetbrains.com/idea/download/?section=windows

Figura 3 Icono de IntelliJ IDEA (versión 2024.3.2.2)

2. Instalar el JDK 17 (Java Development Kit) para desarrollar, compilar y ejecutar aplicaciones en el lenguaje de programación Java, lo podemos encontrar en: https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html

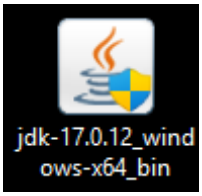


Figura 4. Instalador de JDK 17.0.12 para Windows x64

3. Instalar Git es un sistema de control de versiones para el desarrollo de software, lo podemos encontrar en: https://git-scm.com/downloads/win

Figura 5. Instalador de Git (versión 2.47.1.2 para Windows 64 bits)

4. Instalar GitHub repositorio en la nube donde los desarrolladores pueden trabajar en equipo, hacer seguimiento de cambios y colaborar en proyectos de software. Descargar el Branch del proyecto DevSecOps, lo podemos encontrar en: https://desktop.github.com/download/

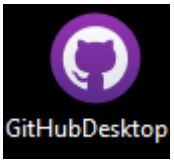


Figura 6 Icono de la Aplicación GitHub Desktop

5. Instalar Docker aplicación que permite ejecutar y gestionar contenedores Docker lo podemos encontrar en: https://www.docker.com/products/docker-desktop/

Figura 7 Instalador de Docker Desktop

6. Instalar Maven es una herramienta de automatización y gestión de proyectos, donde la configuración del proyecto se define en un archivo XML llamado pom.xml. apache-maven-3.9.9 versión específica de Maven, lo podemos encontrar en: https://maven.apache.org/install.html

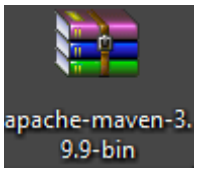


Figura 8 Paquete Binario de Apache Maven (versión 3.9.9)

7. Instalar redis es una base de datos en memoria, lo podemos encontrar en: https://redis.io/downloads/

Figura 9 Logotipo de Redis

8. Instalar Postman facilita la comunicación con servicios web mediante solicitudes HTTP, lo podemos encontrar en: https://www.postman.com/downloads/

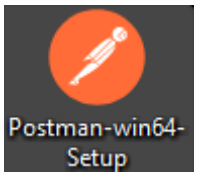


Figura 10. Instalador de Postman para Windows 64 bits

9. Instalar Grafana es una plataforma de visualización y monitoreo de datos como monitoreo de infraestructura y servidores, seguimiento de logs y eventos en aplicaciones y sistemas, análisis de métricas de bases de datos y rendimiento entre otras, lo podemos encontrar en: https://grafana.com/docs/

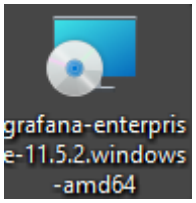


Figura 11. Instalador de Grafana Enterprise (versión 11.5.2 para Windows 64 bits)

10. Instalar Prometheus es un sistema de monitoreo y alerta para visualizar datos en Grafana mediante gráficos y dashboards, lo podemos encontrar en: https://prometheus.io/

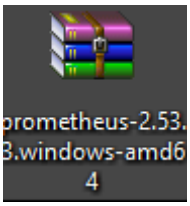


Figura 12. Paquete Binario de Prometheus (versión 2.53.3 para Windows 64 bits)

4.1.3.2. Ambiente en la nube.

Las herramientas que se han configurado para la integración de herramientas de seguridad para microservicios son las siguientes:

4.1.3.2.1 Configuración de secrets en el repositorio de GitHub.

Para configurar los secrets necesarios en el repositorio de GitHub se debe recurrir al repositorio DEVSECOPS/Settings/Secrets and variables/Actions

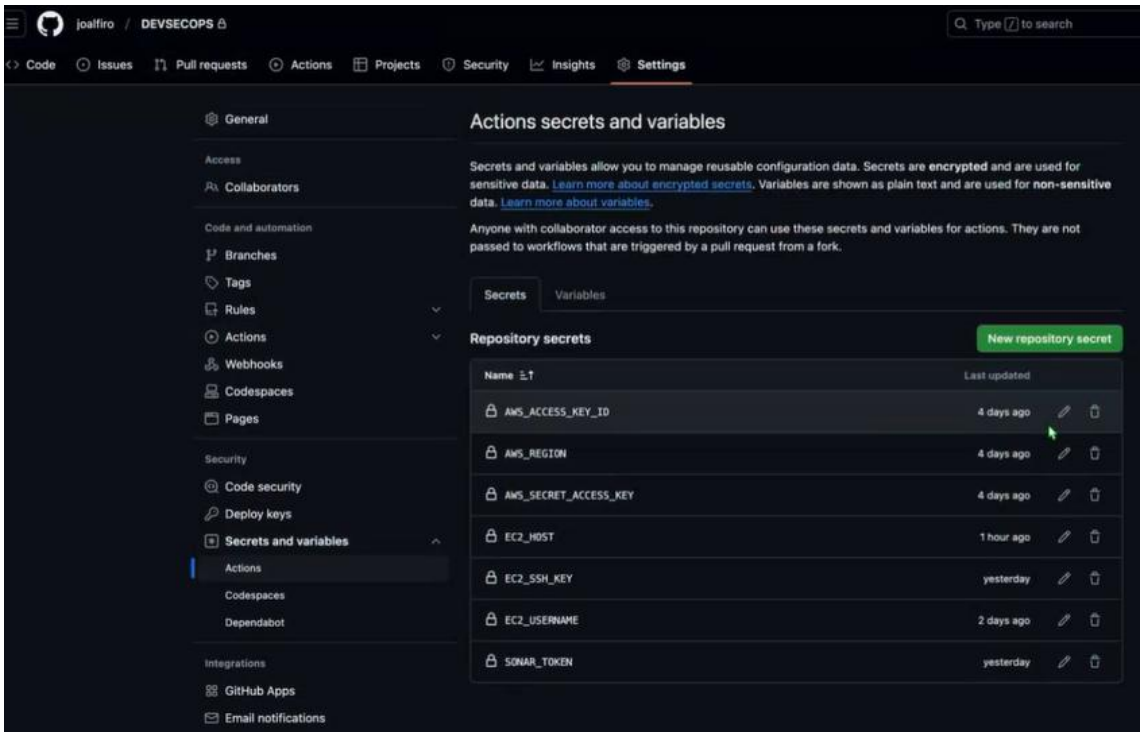


Figura 13 Configuración de Secretos y Variables de Repositorio en GitHub

Los Secrets Necesarios son necesarios para conectar con AWS y se instancie contra la instancia de EC2 en este caso la máquina que está corriendo y permite desplegar la nueva imagen construida con los cambios realizados.

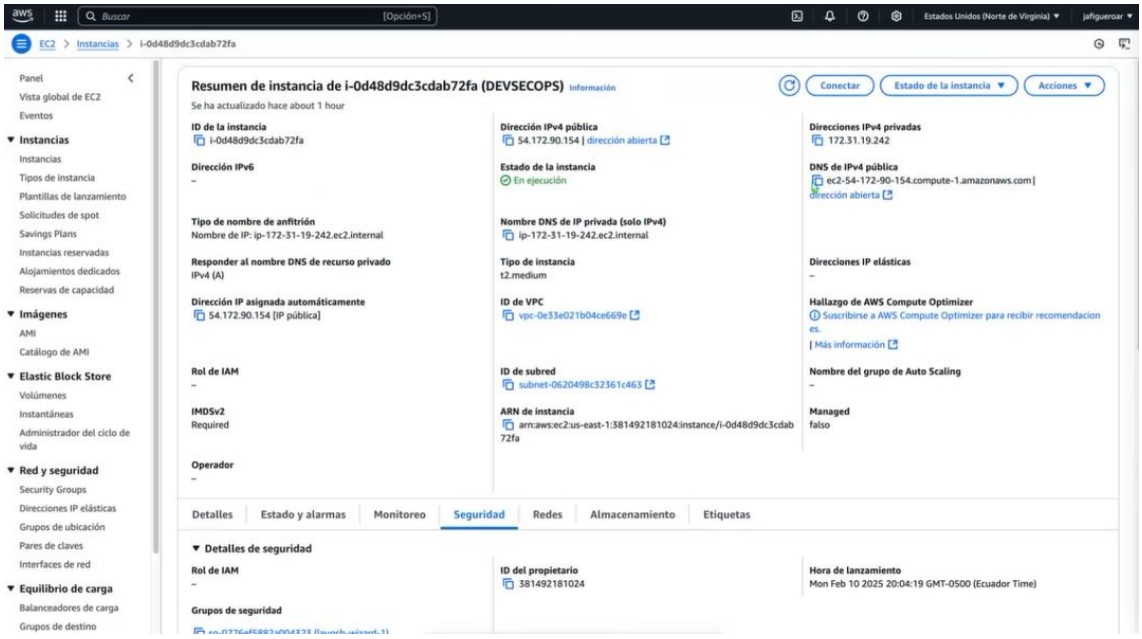


Figura 14. Resumen de la Instancia EC2 DEVSECOPS en AWS

Lo primero que se configura para que funcione son las variables:

AWS_ACCESS_KEY_ID # ID de acceso de AWS

AWS_SECRET_ACCESS_KEY# Clave secreta de AWS

Lo que iría relacionado a la cuenta/credenciales de seguridad

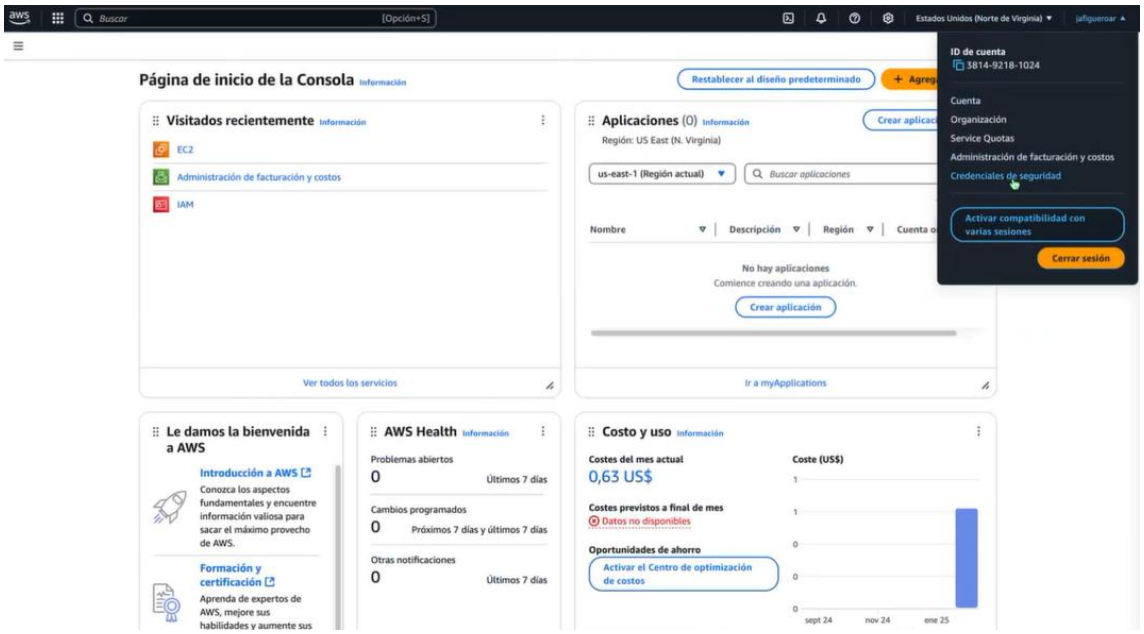


Figura 15 Página de Inicio de la Consola de AWS

En esta sección se ha creado un usuario

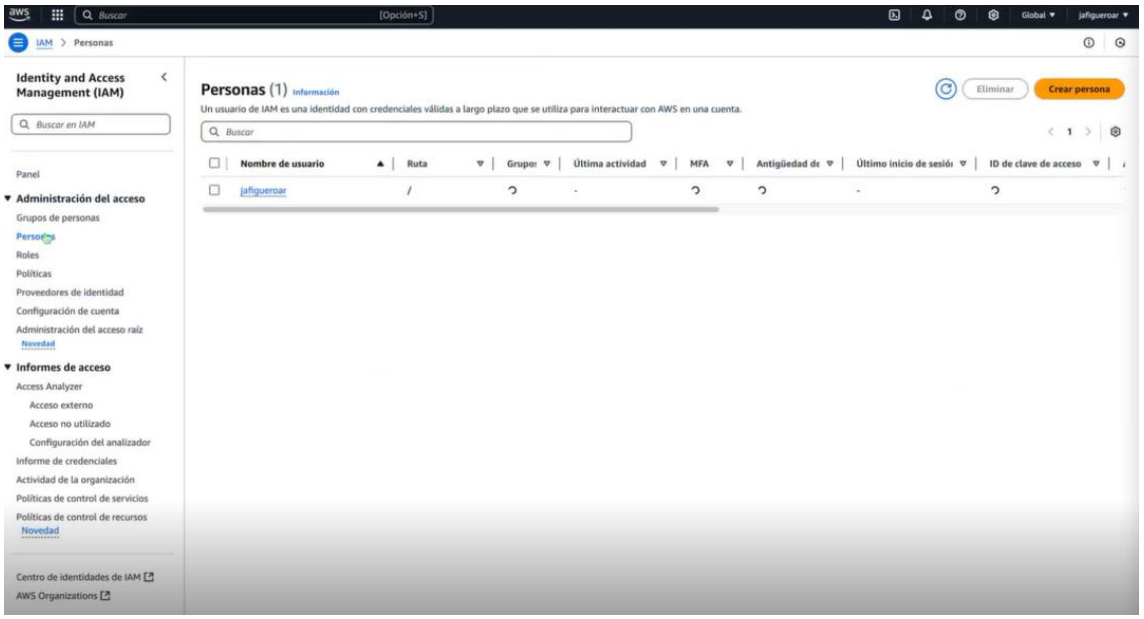


Figura 16 Página de Inicio de la Consola de AWS

Y se ingresa el usuario creado

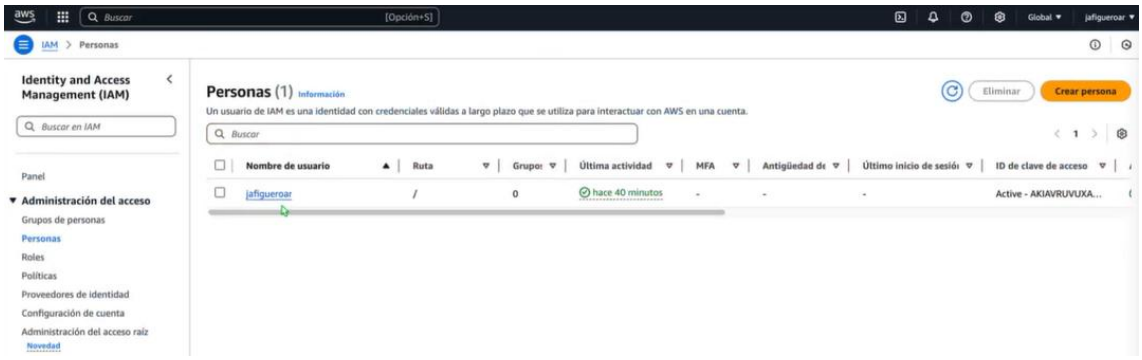


Figura 17 Detalle del Usuario IAM en la Consola de AWS

Se selecciona la opción de credenciales de seguridad

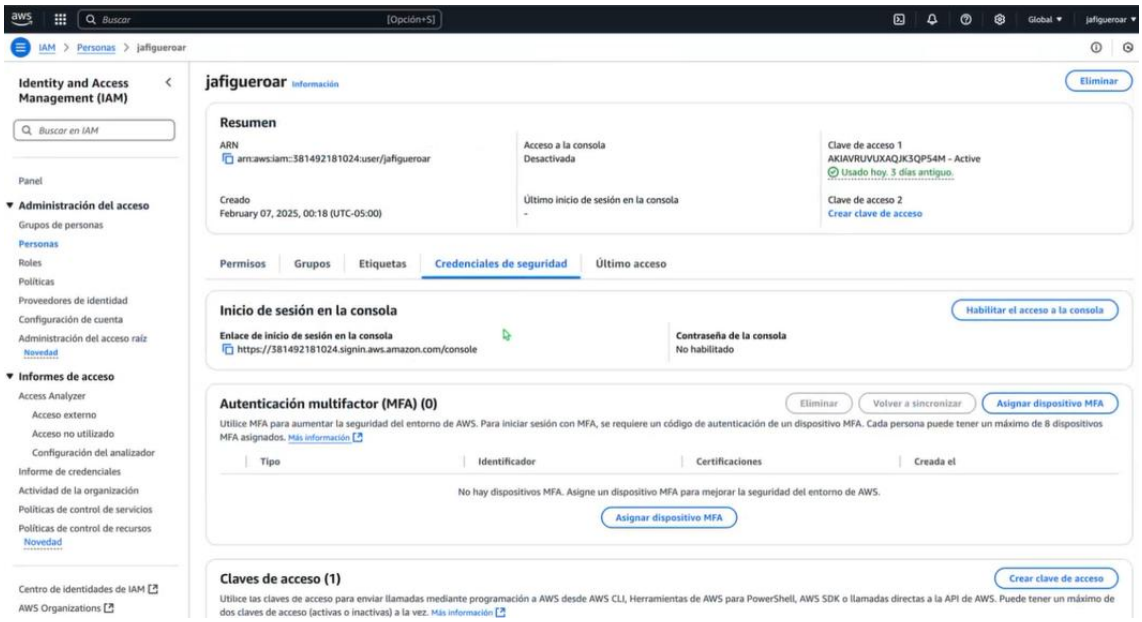


Figura 18 Configuración de Credenciales y MFA para el Usuario IAM

Para crear la clave de acceso se hace clic en Crear clave de acceso

Figura 19 Botón para Crear una Nueva Clave de Acceso en AWS IAM

Se selecciona la Interfaz de línea de comandos (CLI) entiendo la recomendación anterior y deseo proceder a la creación de una clave de acceso y clic en siguiente.

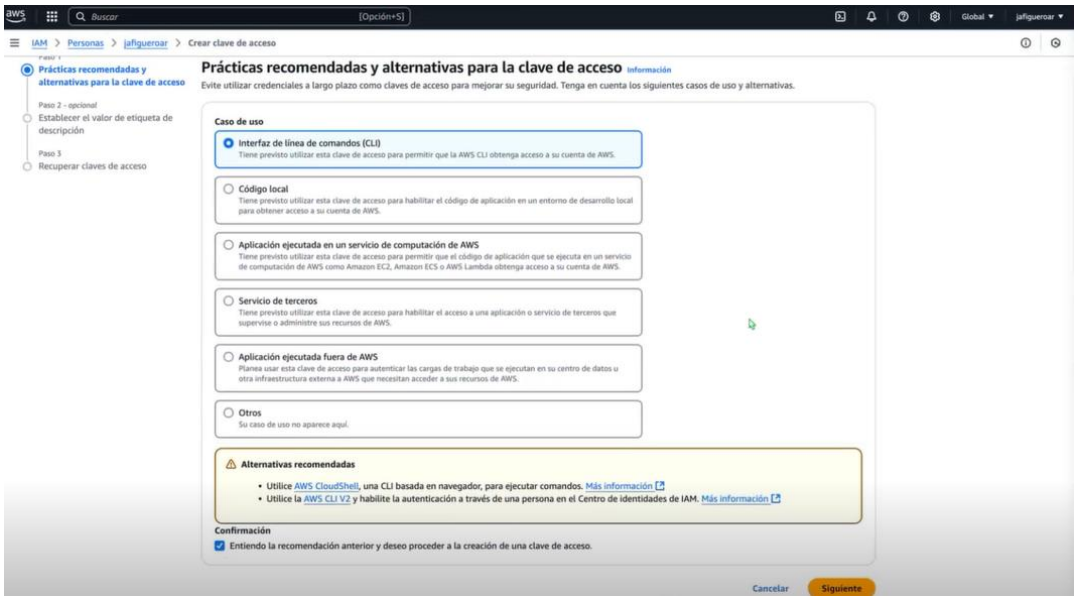


Figura 20 Prácticas Recomendadas y Alternativas para la Clave de Acceso en AWS IAM

En la siguiente pantalla se selecciona crear clave de acceso

Figura 21 Creación de la Clave de Acceso con Etiqueta de Descripción en AWS IAM

Donde se muestra la clave de acceso y la clave de acceso secreta que se muestra por una sola vez

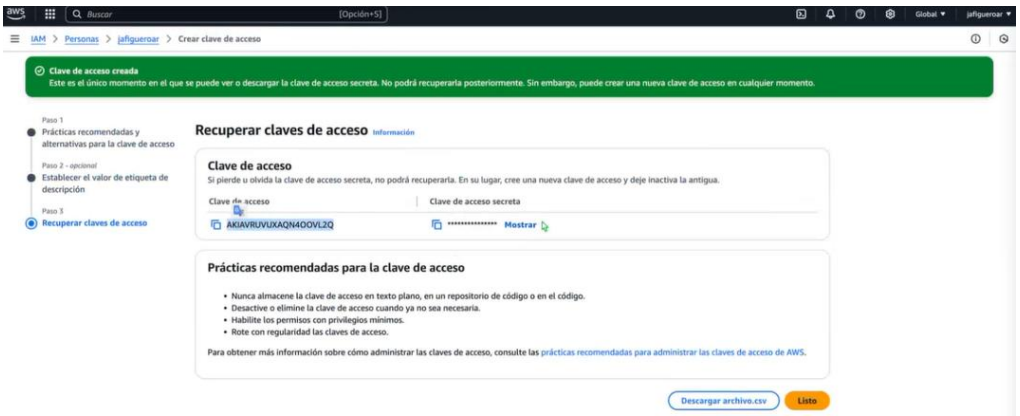


Figura 22. Clave de Acceso Generada y Prácticas Recomendadas en AWS IAM

Aquí se puede observar que ya se encuentra creado una clave de acceso

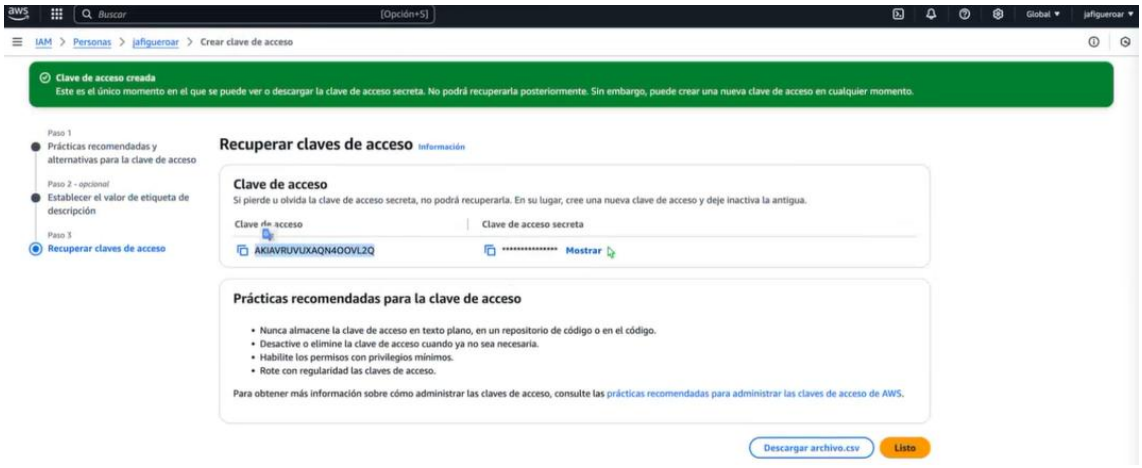


Figura 23 Panel de Claves de Acceso y Claves Públicas SSH en AWS IAM

Además, se configura la variable:

AWS_REGION # Región de AWS

Esta variable depende de donde se cree el recurso, para la práctica se ha utilizado el servicio de Amazon EC2 (Elastic Compute Cloud) un servicio de Amazon Web Services (AWS), que permite a los usuarios alquilar servidores virtuales (instancias) para ejecutar aplicaciones sin necesidad de comprar hardware físico.

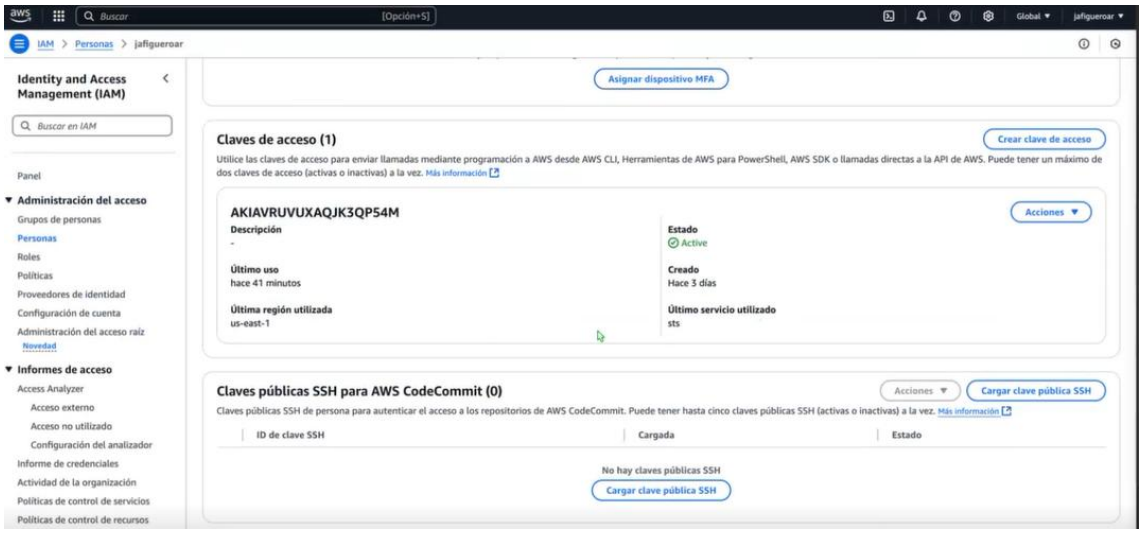


Figura 24 Página de Inicio de la Consola de AWS con la Sección de Aplicaciones

Para crear una instancia se selecciona en el comando Instancias

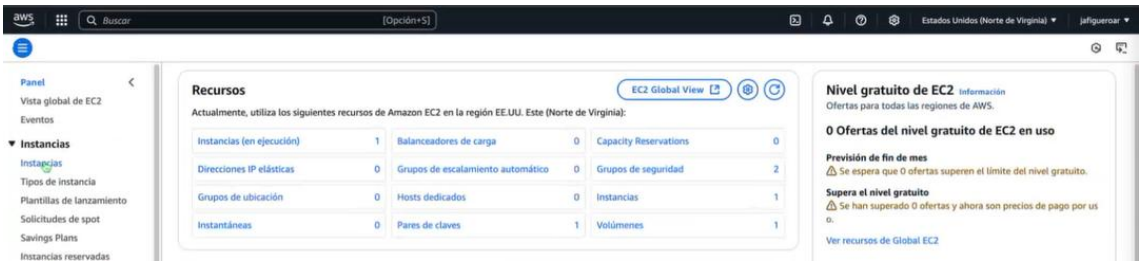


Figura 25 Panel de Instancias en la Consola de EC2 de AWS

Y se procede a dar Clic en lanzar instancias

Figura 26 Botón de Lanzamiento de una Nueva Instancia EC2 en AWS

A continuación, se ingresa el nombre: DEVSECOPS

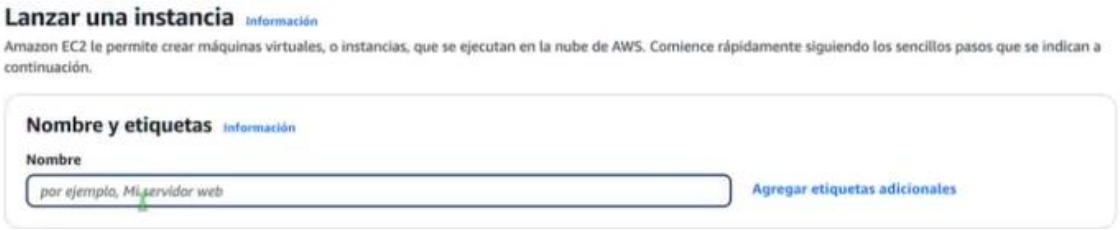


Figura 27 Proceso de Lanzamiento de una Nueva Instancia EC2 en AWS

Se selecciona, además, el sistema operativo Amazon Linux 2023 y la arquitectura de 64 bits.

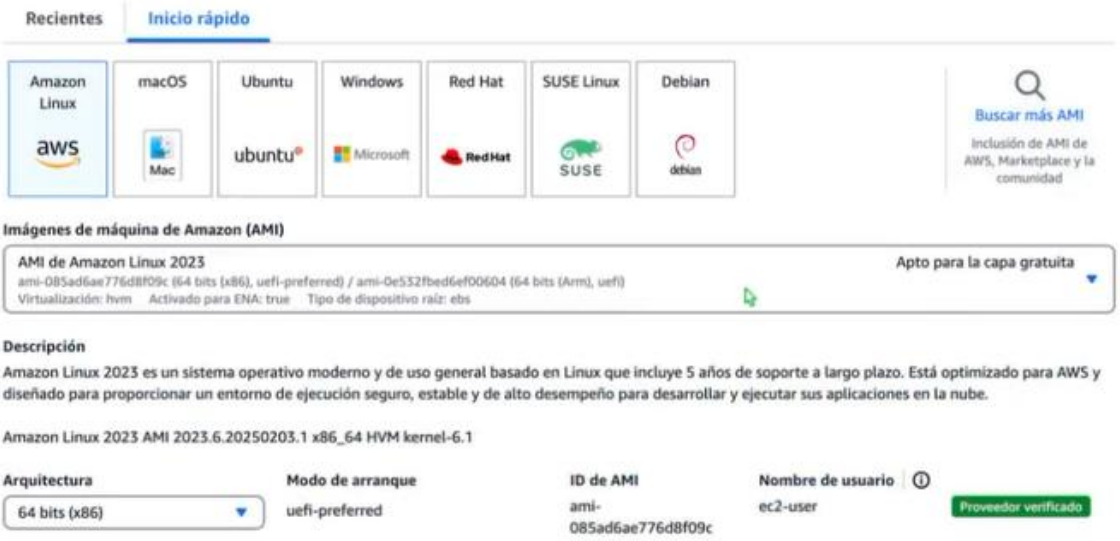


Figura 28. Selección de AMI Amazon Linux 2023 para Instancia EC2

Para Docker se necesitan mínimo 4 GB en ram para esto se selecciona el tipo de instancia t2.medium

Figura 29. Selección del Tipo de Instancia t2.medium en EC2

La pantalla muestra la clave privada ssh creada que es la que está en EC2_SSH_KEY y que se asocia a la máquina.

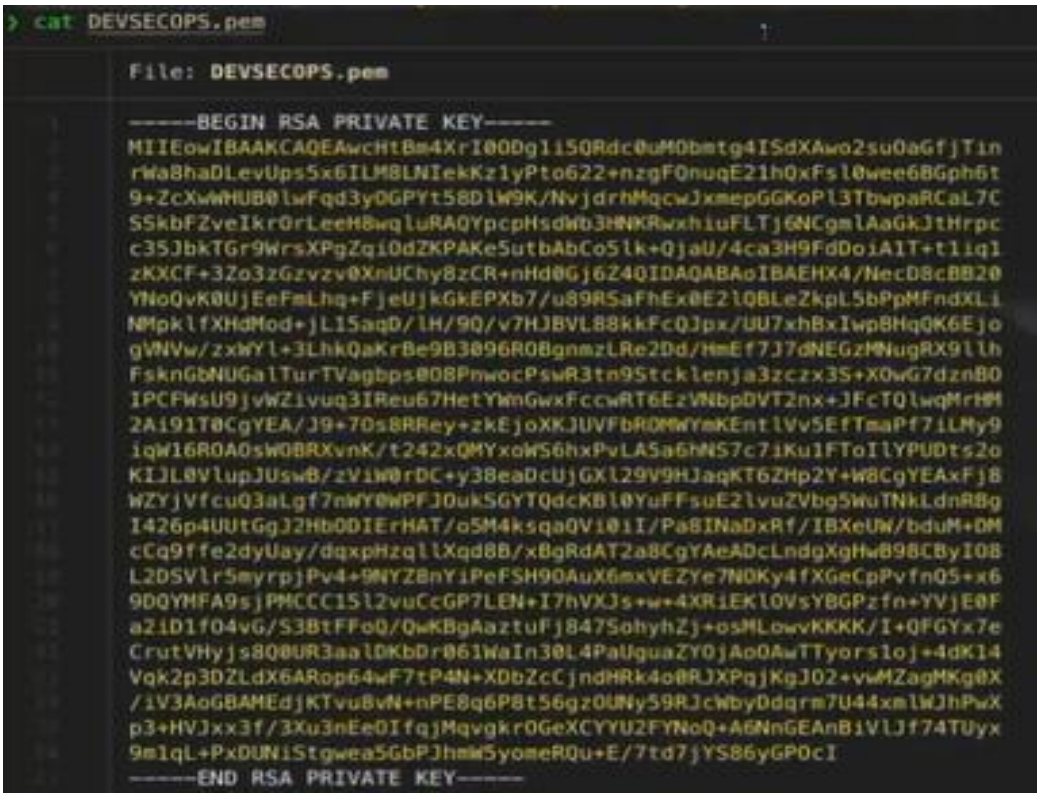


Figura 30. Visualización del Archivo DEVSECOPS.pem

Se selecciona en DEVSECOPS tipo:rsa

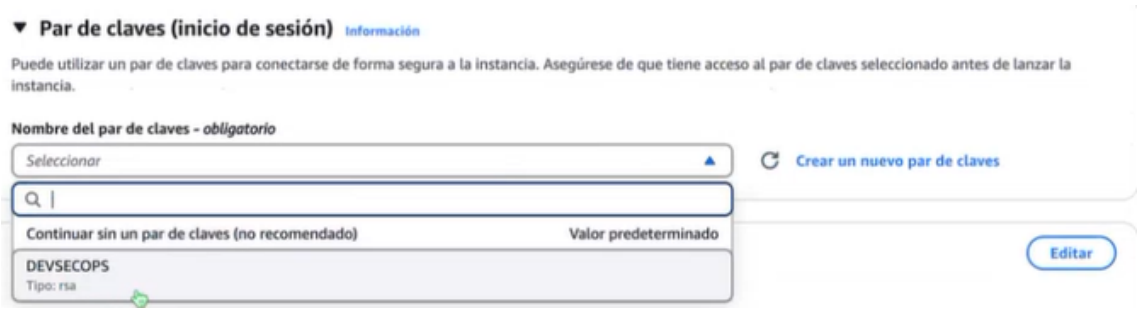


Figura 31. Selección del Par de Claves DEVSECOPS para la Instancia EC2

En configuraciones de red se debe seleccionar: crear grupo de seguridad, permitir el tráfico SSH desde, cualquier lugar para que permita conectarse a la máquina de AWS.

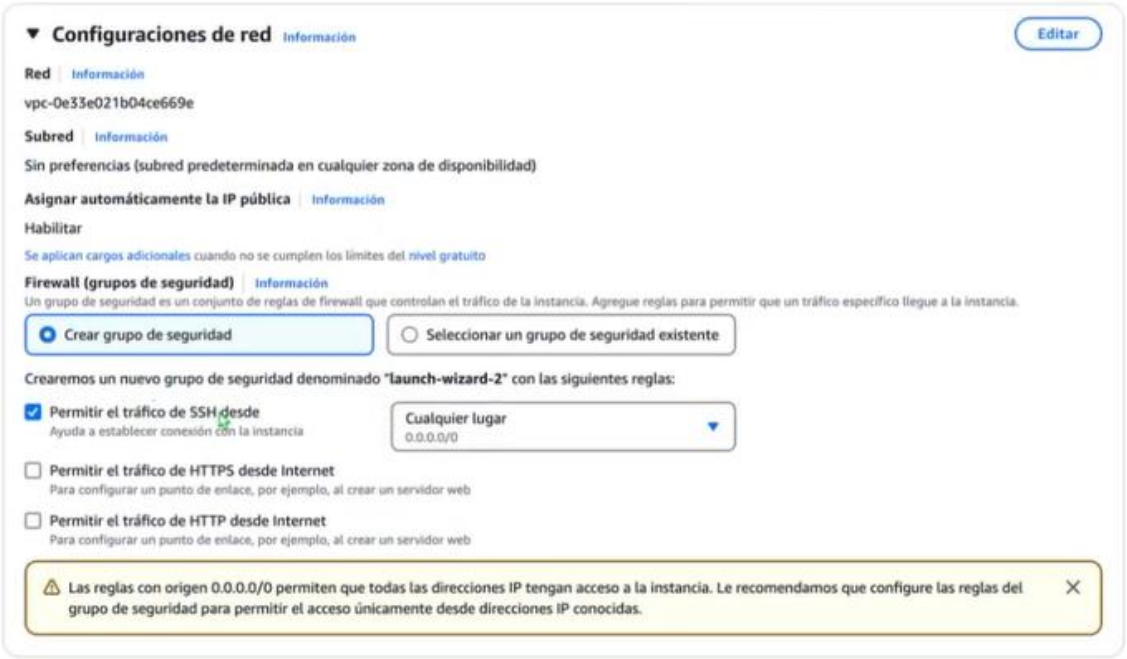


Figura 32. Configuración de Red y Reglas de Seguridad para la Instancia EC2

En configurar almacenamiento se necesitan 8 GB adicionales.

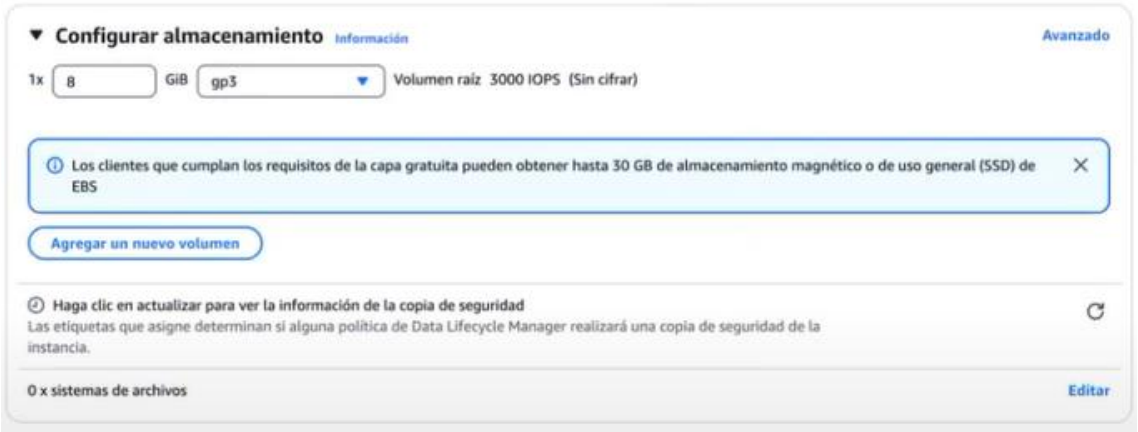


Figura 33. Configuración de Almacenamiento para la Instancia EC2

Finalmente, se realiza un clic en lanzar instancia

Figura 34. Resumen de Configuración y Lanzamiento de la Instancia EC2

Se puede observar el detalle de la instancia creada.

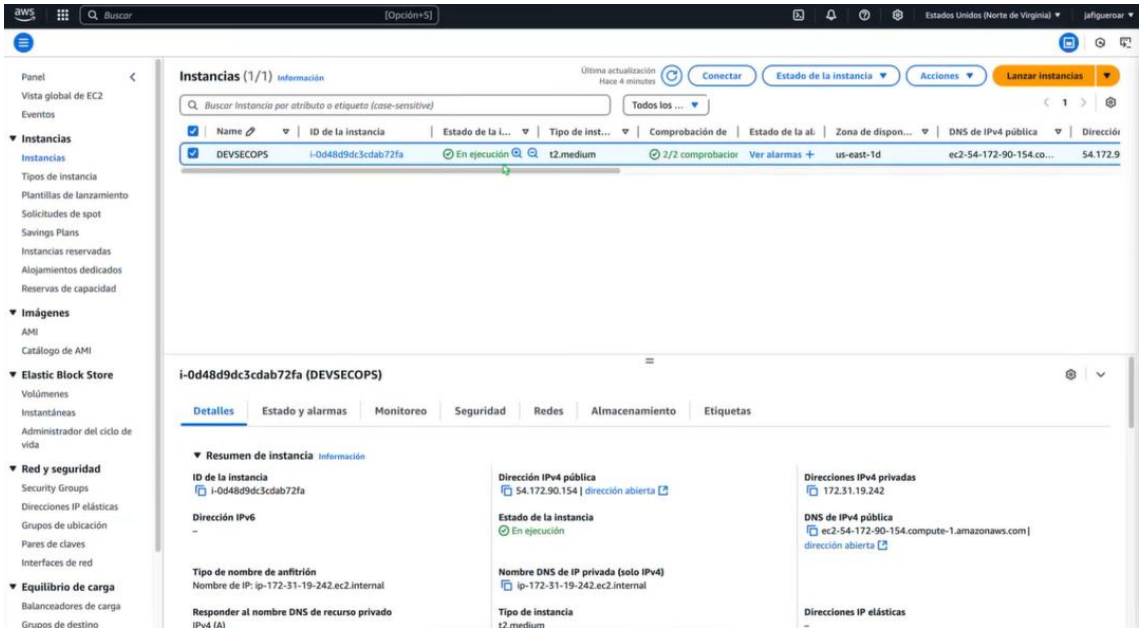


Figura 35. Instancia EC2 “DEVSECOPS” Ejecutándose en AWS

Lo anterior se utiliza para configurar la variable EC2_HOST # IP o DNS de la instancia EC2 (host remoto de una ip pública). En la pantalla se observa la instancia creada DEVSECOPS y los detalles de la instancia creada.

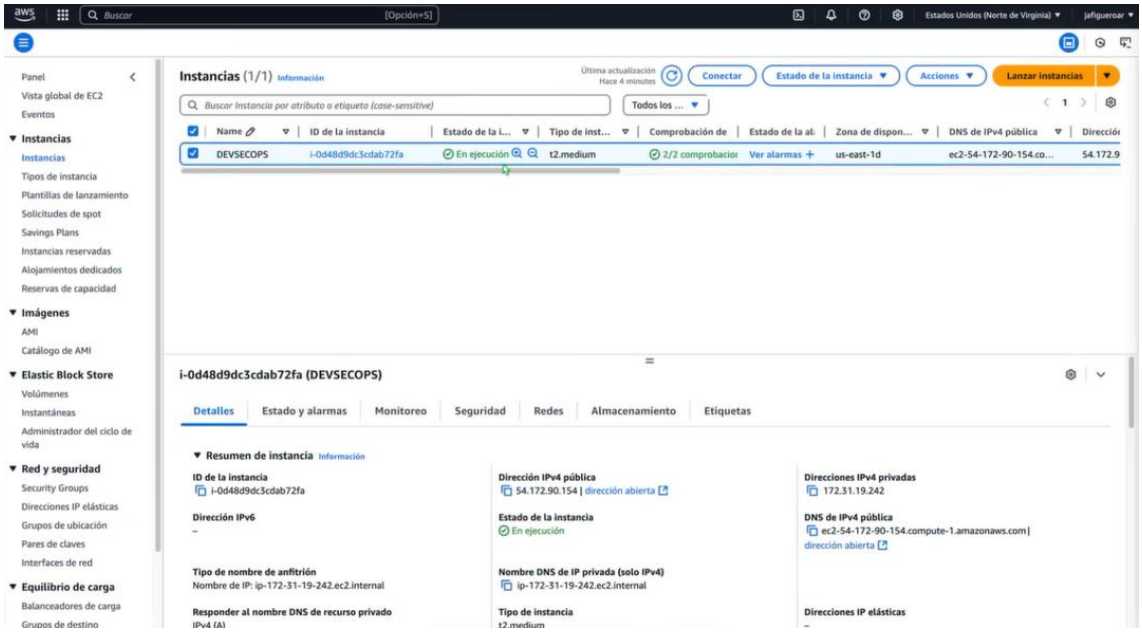


Figura 36. Panel de Información Detallada de la Instancia EC2 "DEVSECOPS" en AWS

Para la práctica se ha utilizado una ip pública dinámica es decir que se apaga la instancia creada en Detener instancia

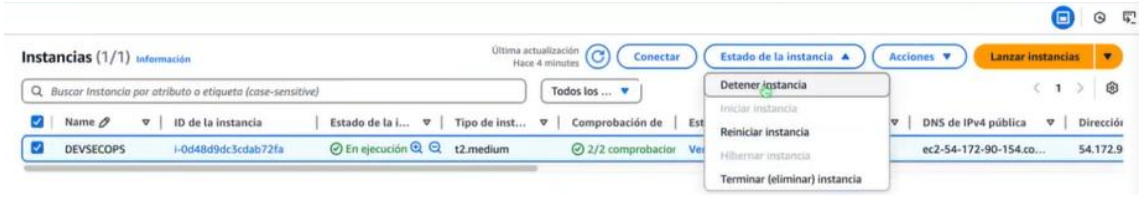


Figura 37. Opciones de Gestión de la Instancia EC2 "DEVSECOPS" en AWS

Para ello se realiza clic en Detener

Figura 38. Detención de la Instancia EC2 “DEVSECOPS” en AWS

Sucede lo siguiente, se cierra la conexión ssh de la máquina

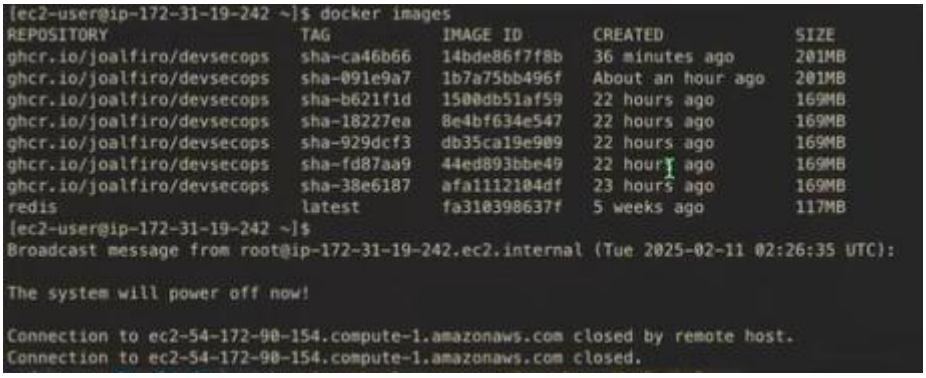


Figura 39. Listado de Imágenes Docker con Etiquetado SHA en la Instancia EC2

Se vuelve a iniciar instancia y encender

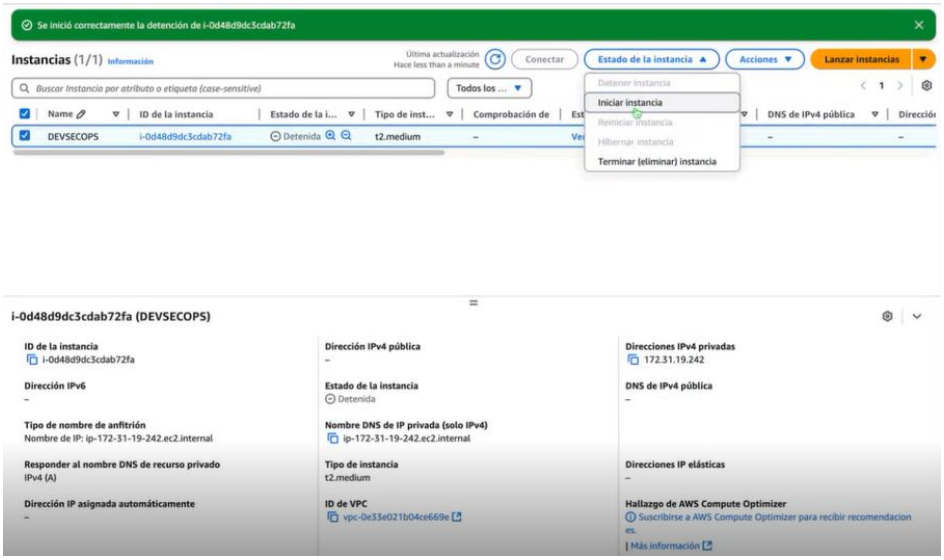


Figura 40. Inicio de la Instancia EC2 “DEVSECOPS” en la Consola de AWS

Se observa que se ha realizado el cambio a ip pública

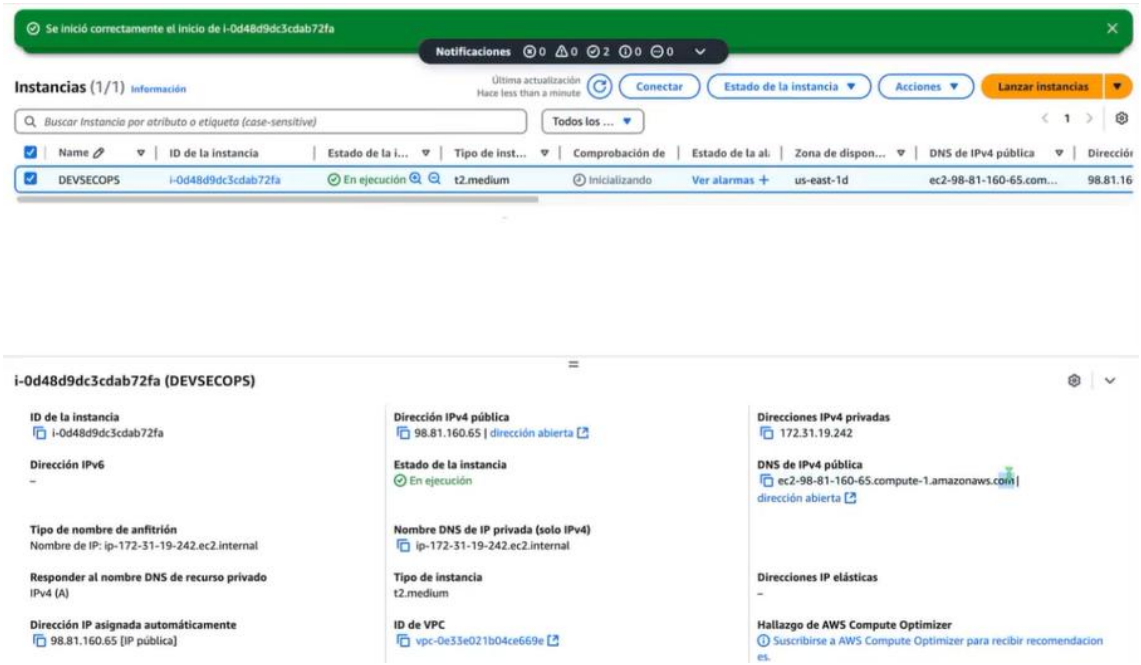


Figura 41. Mensaje de Confirmación al Iniciar la Instancia EC2 “DEVSECOPS” en AWS

Para esto se copia

Figura 42. Copia de la DNS IPv4 Pública de la Instancia EC2 en AWS

Se procede a dirigirse a GitHub a la sección de Secrets and variables/Actions en EC2_HOST

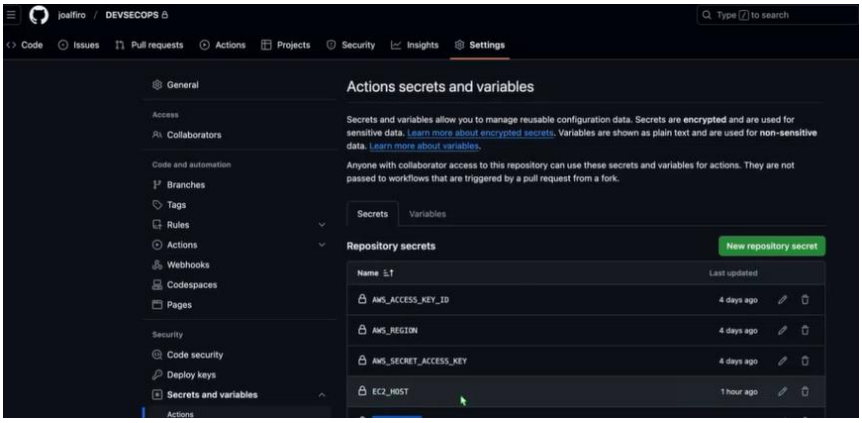


Figura 43. Configuración de Secrets y Variables de Entorno en GitHub

Se selecciona el lápiz para pegar el DNS IPv4 público y clic en Update secret para que se actualice el nuevo host.

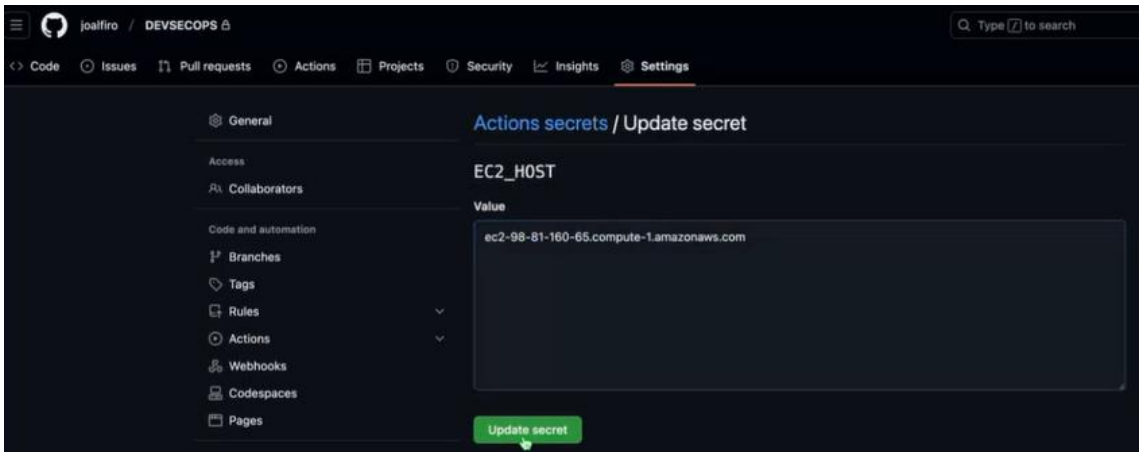


Figura 44. Actualización del Secreto EC2_HOST en GitHub Actions

Desde la conexión ssh de la máquina cliente ya no se puede conectar a la anterior versión porque la ip ya no existe para esto se selecciona el comando Conectar

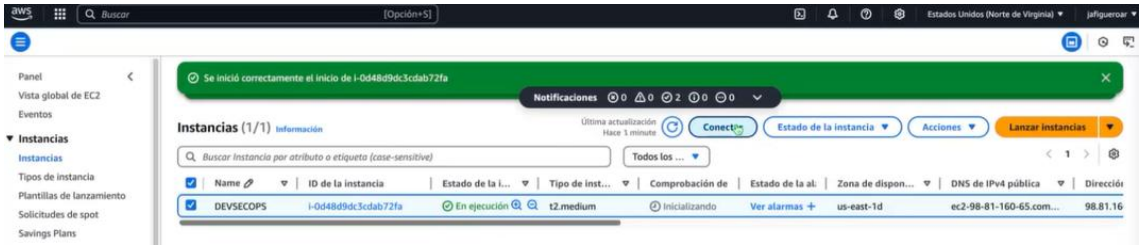


Figura 45. Notificación de Inicio de la Instancia EC2 “DEVSECOPS” en la Consola de AWS

En Cliente SSH se copia lo que se encuentra seleccionado

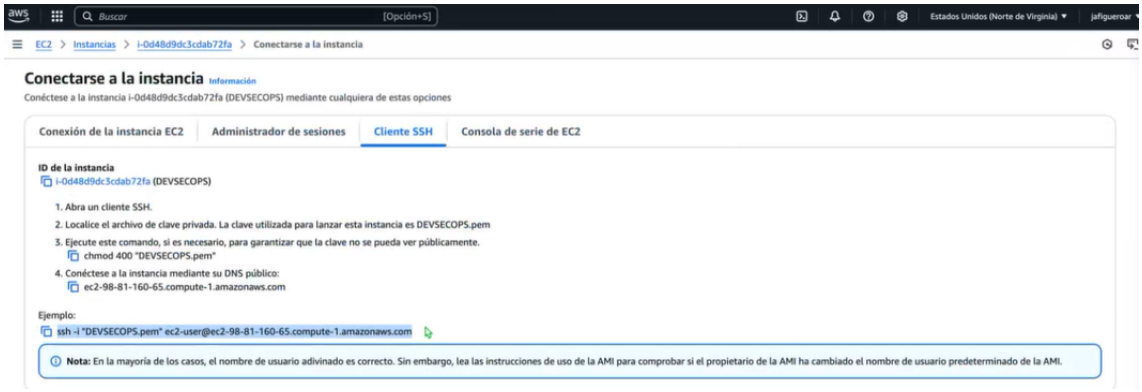


Figura 46. Instrucciones para Conectarse a la Instancia EC2 “DEVSECOPS” mediante SSH

En la conexión ssh se pega el Cliente SSH y se verifica que se encuentra corriendo la aplicación

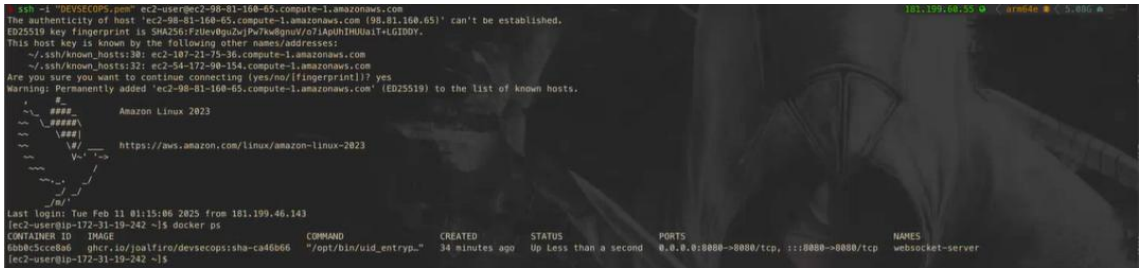


Figura 47. Conexión SSH a la Instancia EC2 “DEVSECOPS” y Visualización de Contenedores Docker

Para configurar la variable EC2_USERNAME # Usuario SSH de EC2 (ssh de la clave privada), en GitHub a la sección de Secrets and variables/Actions en EC2_USERNAME

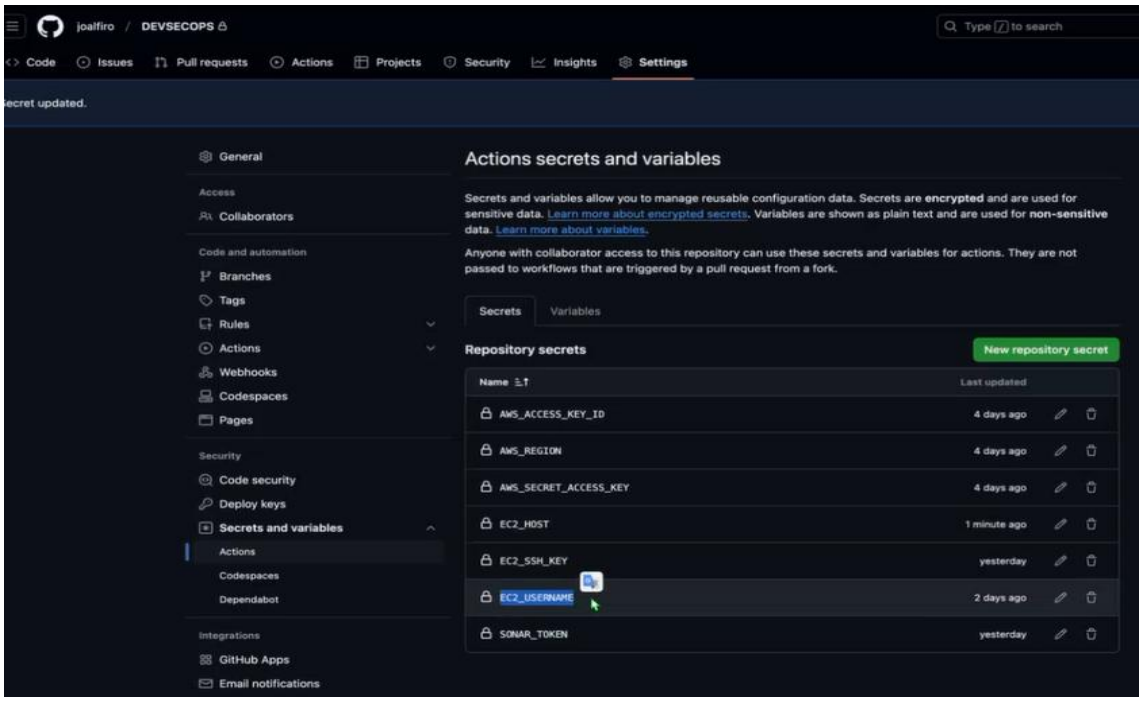


Figura 48. Visualización y Actualización de Secrets en GitHub Actions

En Cliente SSH se copia lo seleccionado y se actualiza la variable EC2_USERNAME de GitHUb

Figura 49. Opciones de Conexión SSH a la Instancia EC2 “DEVSECOPS” en la Consola de AWS

Para configurar la variable SONAR_TOKEN # Token de Sonar (automáticamente proporcionado), se conecta GitHub con SonarQube en

https://sonarcloud.io/explore/projects, para esto se crea una organización que contiene el proyecto: DEVSECOPS, en branches se verifica que la versión gratuita solo permite monitorear la rama principal. El servicio en la nube de SonarQube tiene limitaciones de acuerdo a la cantidad de ramas que se puedan analizar ya que solo permite validaciones del branch principal en la versión gratuita, sin embargo, SonarQube puede analizar todas las ramas del proyecto siempre y cuando se haga un upgrade del plan gratuito.

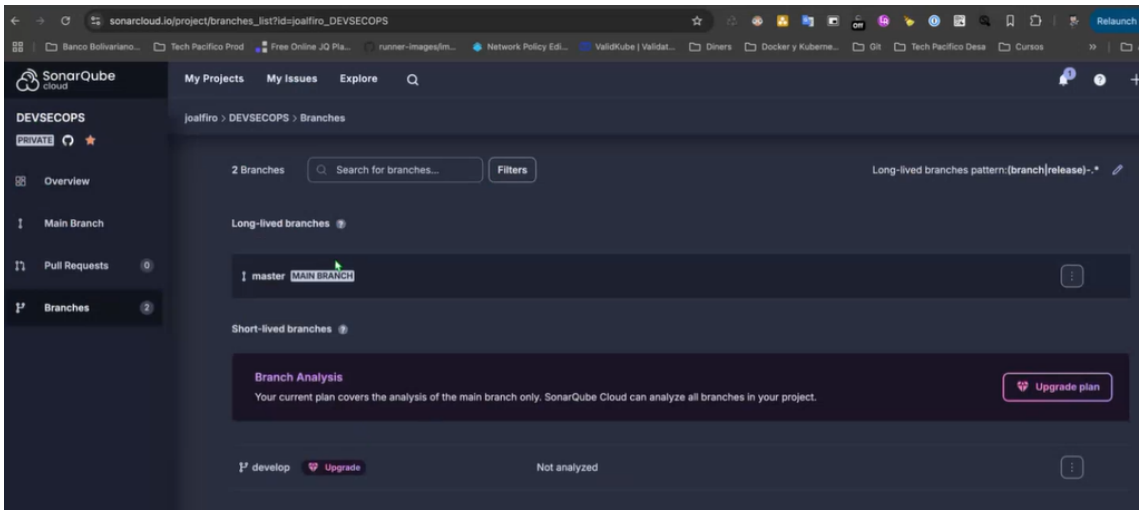


Figura 50. Análisis de Ramas del Proyecto “DEVSECOPS” en SonarQube

Para esto en GitHub dirigirse a Pull requests y seleccionar New pull resquest

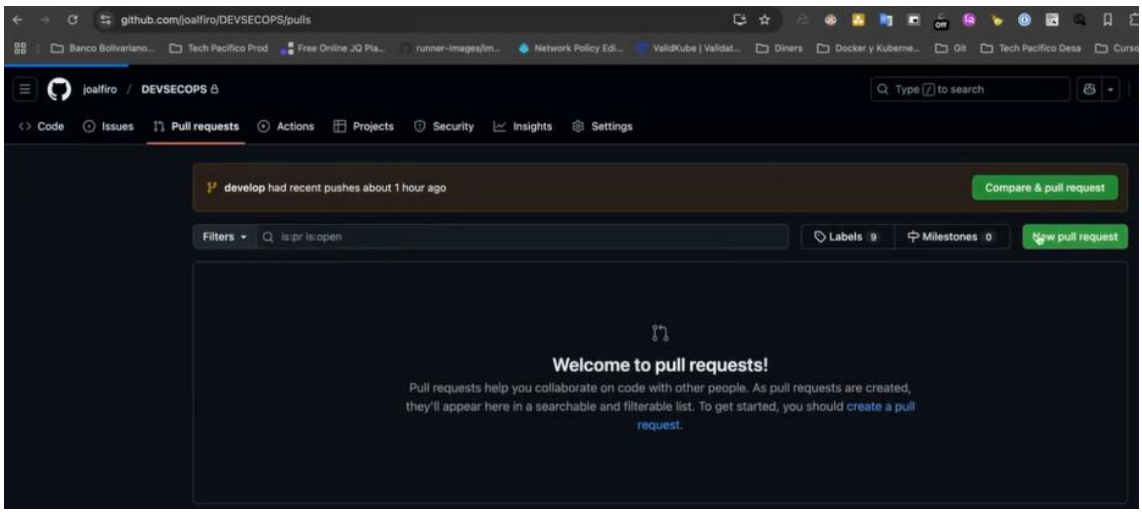


Figura 51. Visualización de Pull Requests en el Repositorio “DEVSECOPS” en GitHub

Se selecciona base: master y en compare: develop

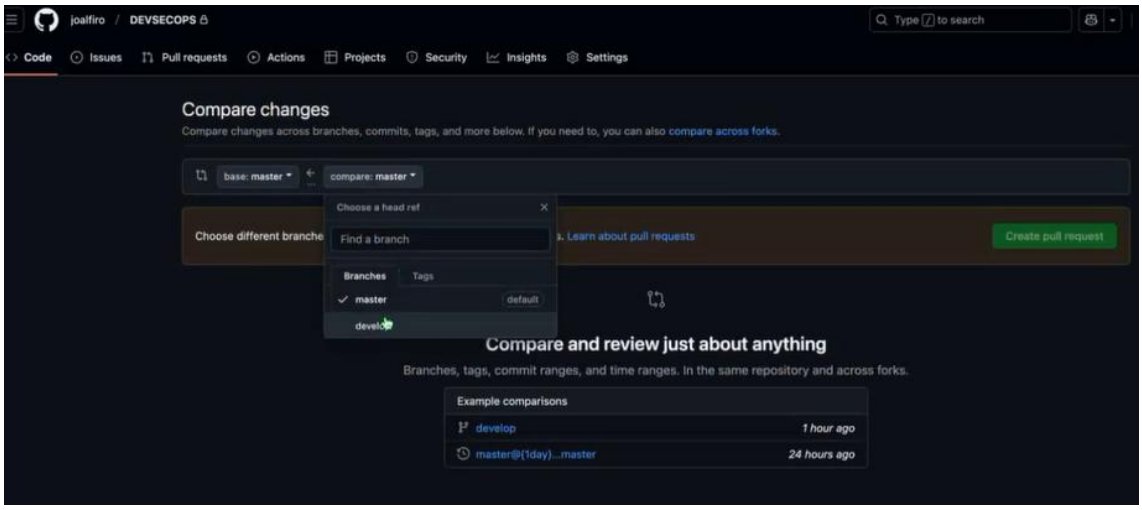


Figura 52. Comparación de Cambios entre las Ramas “master” y “develop” en GitHub

Después se debe hacer clic en Create pull requests

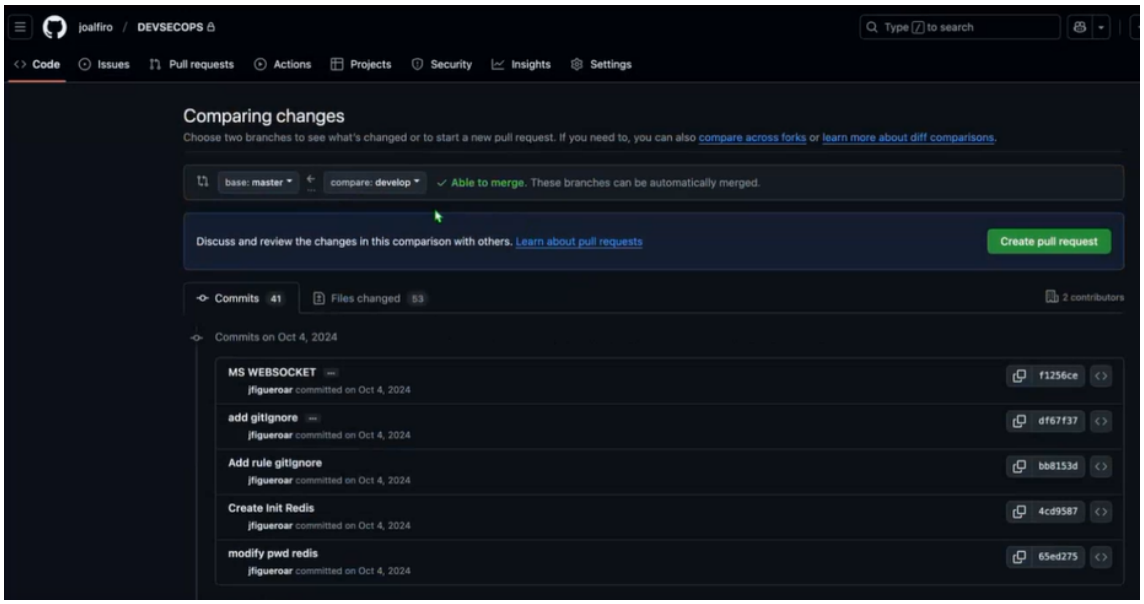


Figura 53. Comparación de Cambios entre “master” y “develop” con Opción de Merge Automático en GitHub

Finalmente, se agrega el título: Develop a master y clic en Create pull requests

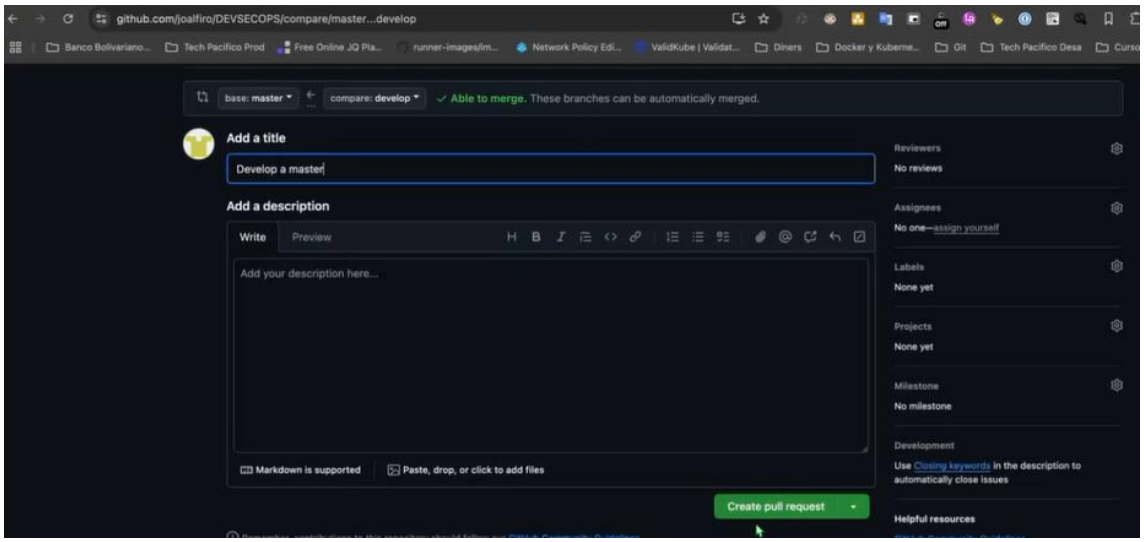


Figura 54. Creación de Pull Request para Fusionar la Rama “develop” con “master” en GitHub

Se configura el pipeline para master para esto se ingresa en DEVSECOPS/.github/workflows/docker-build-deploy.yml y clic en Commit Changes

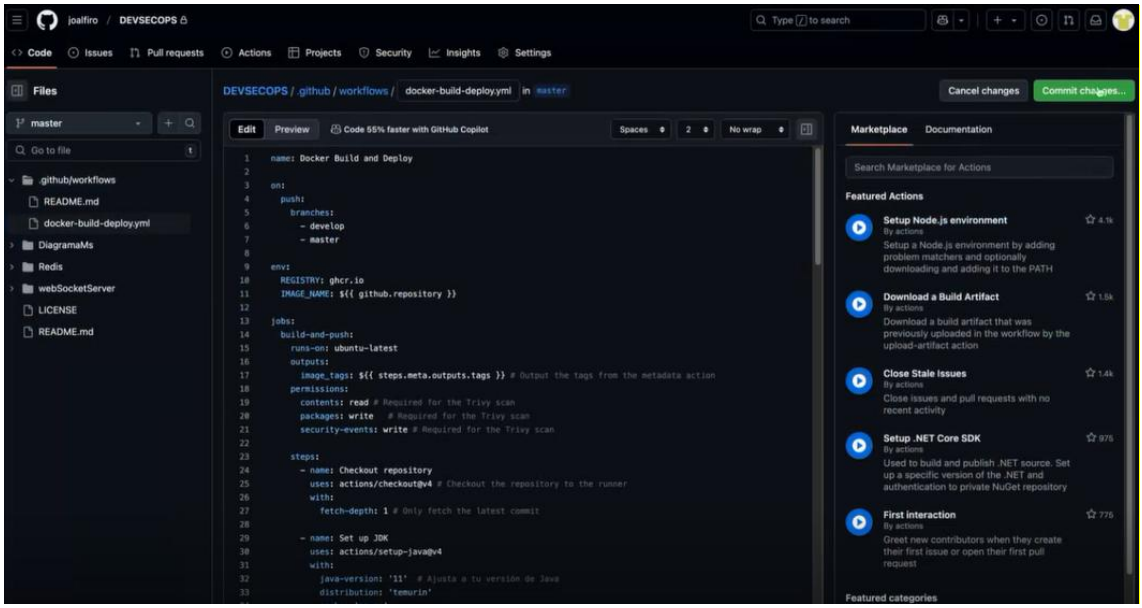


Figura 55. Edición del Archivo docker-build-deploy.yml en GitHub Actions

Y de debe hacer clic en Saving

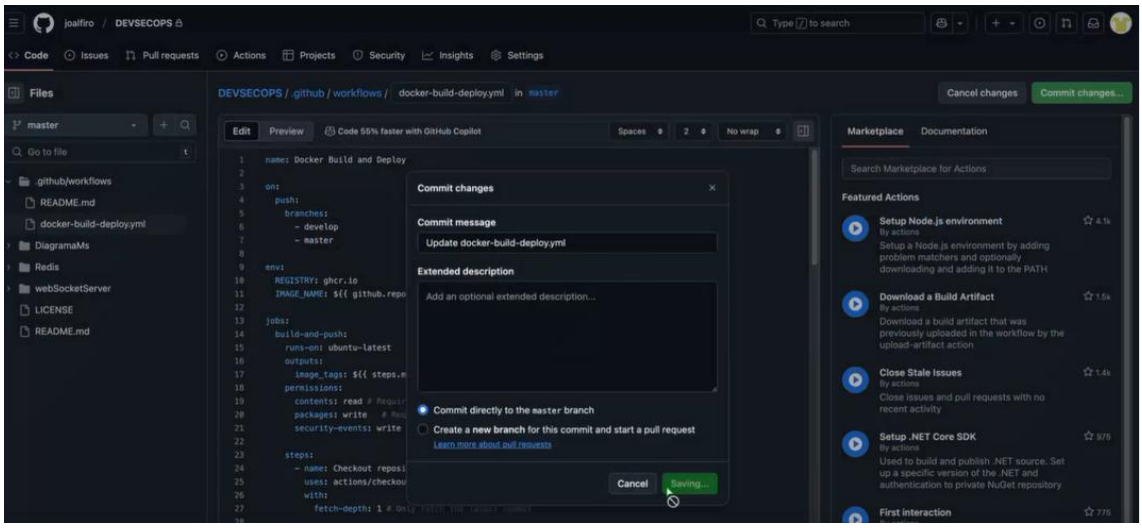


Figura 56. Confirmación de Cambios en el Archivo docker-build-deploy.yml Directamente en GitHub

Finalmente, se verifica la actualización realizada

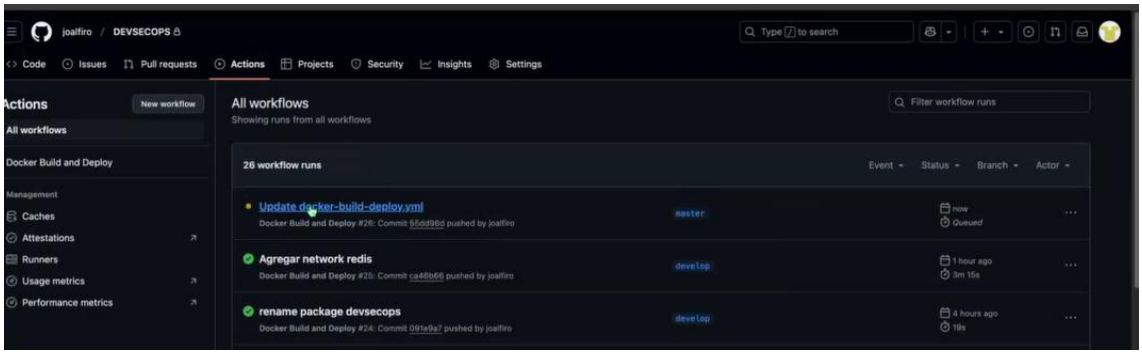


Figura 57. Historial de Ejecuciones de Workflows en GitHub Actions

La variable SONAR_TOKEN se obtiene en sonarQube/jalfiro/security

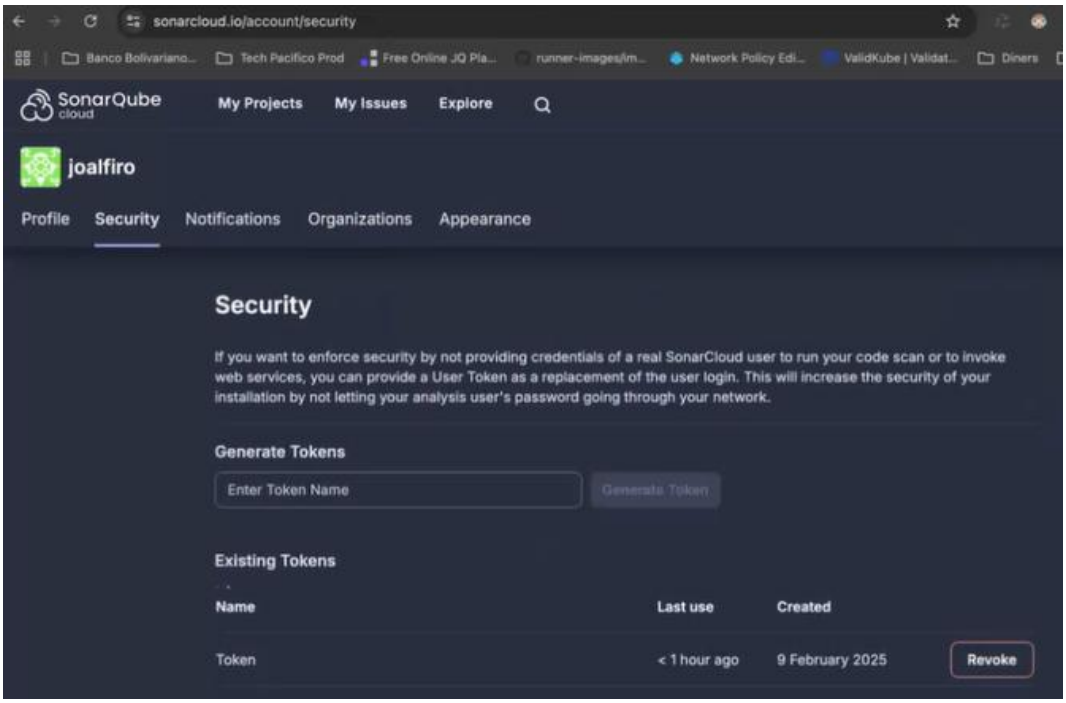


Figura 58. Configuración de Tokens de Seguridad en SonarCloud

4.1.3.3. Configuración de pipeline

La configuración del pipeline se encuentra en DEVSECOPS/.github/workflows/docker-build-deploy.yml, el paso a paso del proceso se visualiza en la figura a continuación. Cada vez que se realice un push en develop o master se va a activar

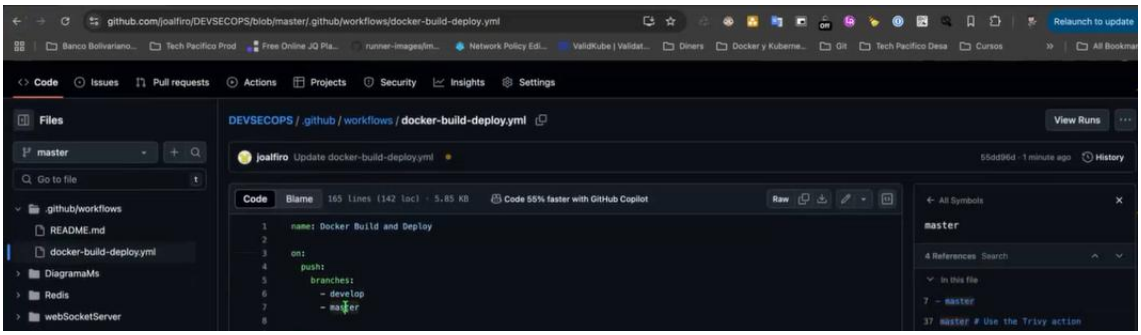


Figura 59. Archivo docker-build-deploy.yml en el Repositorio GitHub

La variable REGISTRY en la imagen está compuesta por ghcr (GitHub conteiner register), encambio, IMAGE_NAME es la variable reservada de GitHub que contiene nombre del usuario más el nombre del repositorio.

```yaml
env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}
```

Figura 60. Variables de Entorno Definidas en el Archivo docker-build-deploy.yml

Posterior se indica el Linux a utilizar.

```yaml
jobs:
  build-and-push:
    runs-on: ubuntu-latest
    outputs:
      image_tags: ${{ steps.meta.outputs.tags }} # Output the tags from the metadata action
    permissions:
      contents: read # Required for the Trivy scan
      packages: write   # Required for the Trivy scan
      security-events: write # Required for the Trivy scan
```

Figura 61. Definición del Job “build-and-push” en el Workflow docker-build-deploy.yml

Además se indica la salida outputs: la variable image_tags donde se guardará los steps.meta.outputs.tags

```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v4 # Checkout the repository to the runner
    with:
      fetch-depth: 1 # Only fetch the latest commit

  - name: Set up JDK
    uses: actions/setup-java@v4
    with:
      java-version: '17'  # Ajusta a tu versión de Java
      distribution: 'temurin'
      cache: 'maven'

  - name: Run Trivy vulnerability scanner (Filesystem scan)
    uses: aquasecurity/trivy-action@master # Use the Trivy action
    with:
      scan-type: 'fs'
      scan-ref: 'webSocketServer'
      format: 'table'
      severity: 'CRITICAL,HIGH'
      output: 'trivy-results.txt'
```

Figura 62. Pasos del Workflow para Configurar Java y Ejecutar Trivy en GitHub Actions

En la siguiente figura se puede visualizar la información de meta

```yaml
- name: Extract metadata for Docker
  id: meta
  uses: docker/metadata-action@v5 # Extract metadata for the Docker image
  with:
    images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
    tags: |
      type=sha,prefix=sha-

- name: Build and push Docker image
  uses: docker/build-push-action@v5 # Build and push the Docker image
  with:
    context: ./webSocketServer
    file: ./webSocketServer/Dockerfile
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
```

Figura 63. Extracción de Metadatos y Construcción de la Imagen Docker en GitHub Actions

Seguidamente, se visualiza la información de tags derivada del prefijo sha de Docker para posterior utilizarla en la compilación

```yaml
- name: Extract metadata for Docker
  uses: docker/metadata-action@v5 # Extract metadata for the Docker image
  with:
    images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
    tags: |
      type=sha,prefix=sha-

- name: Build and push Docker image
  uses: docker/build-push-action@v5 # Build and push the Docker image
  with:
    context: ./webSocketServer
    file: ./webSocketServer/Dockerfile
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
```

Figura 64. Pasos de Metadatos y Construcción de la Imagen Docker en GitHub Actions

Cuando se realiza el commit details se muestra el número

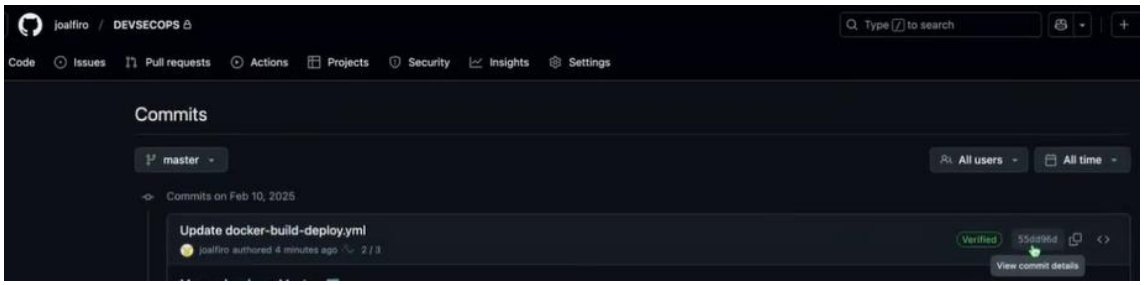


Figura 65. Visualización de Commits y Detalles en el Repositorio “DEVSECOPS” en GitHub

Este número es el que te toma para generar la imagen

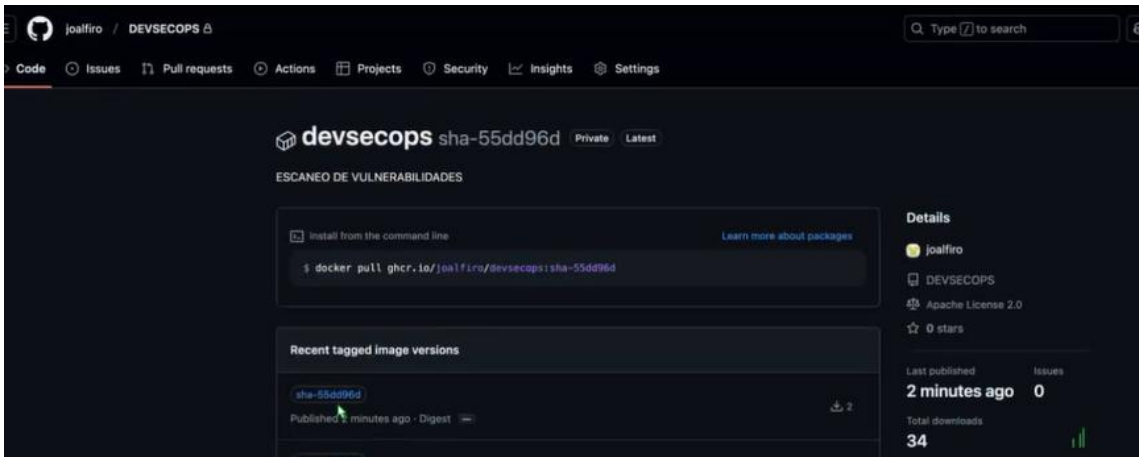


Figura 66. Imagen Etiquetada “sha-55dd96d” en GitHub Container Registry

Posteriormente se debe verificar build-and-push

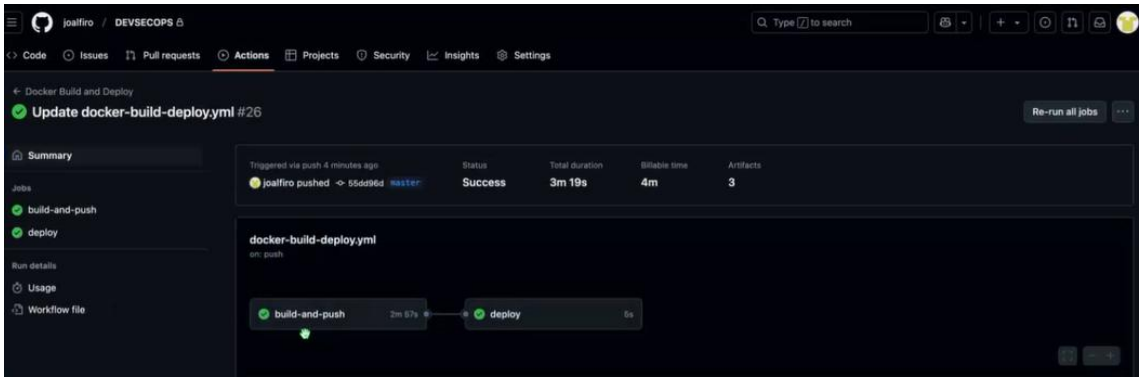


Figura 67. Ejecución Exitosa del Workflow “docker-build-deploy.yml” en GitHub Actions

Se verifica también Sonar

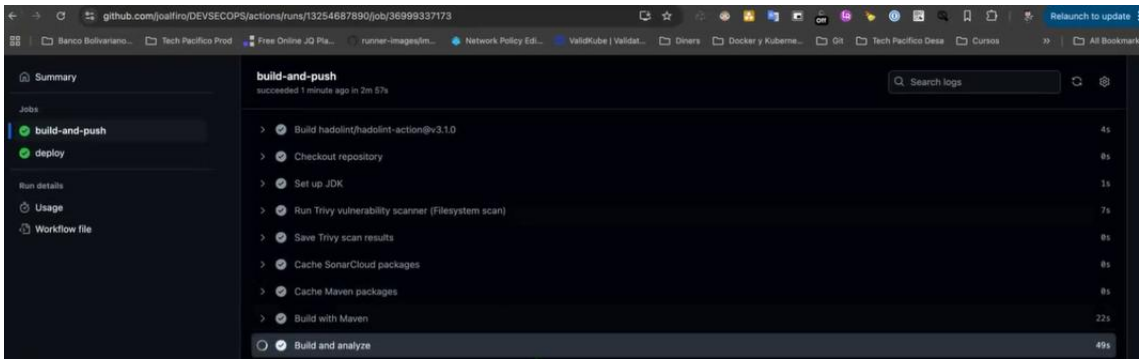


Figura 68. Detalles de los Pasos del Job “build-and-push” en GitHub Actions

Se ingresa para ello en el link para verificar SonarQube

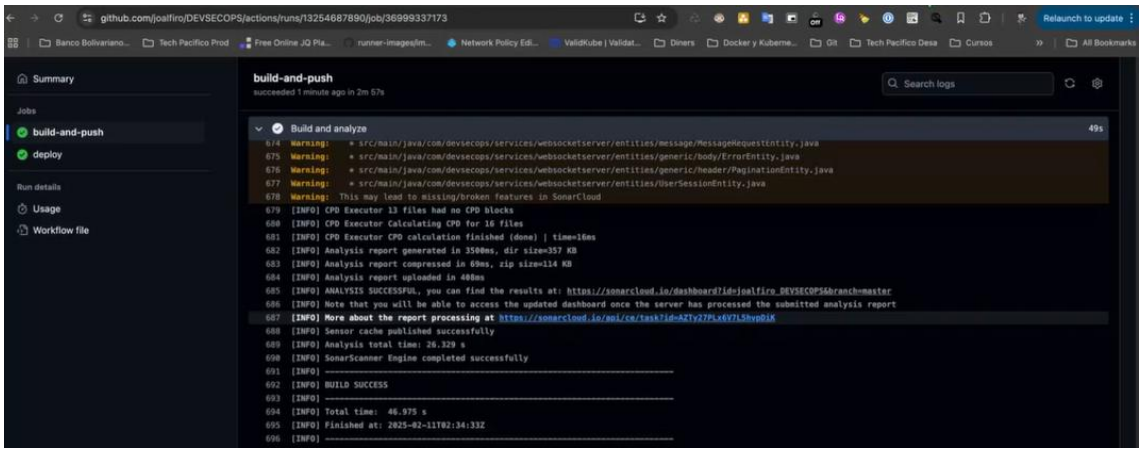


Figura 69. Logs de la Etapa “Build and Analysis” en GitHub Actions

Y es allí donde se muestra la siguiente pantalla del análisis de SonarQube cloud con Github Actions

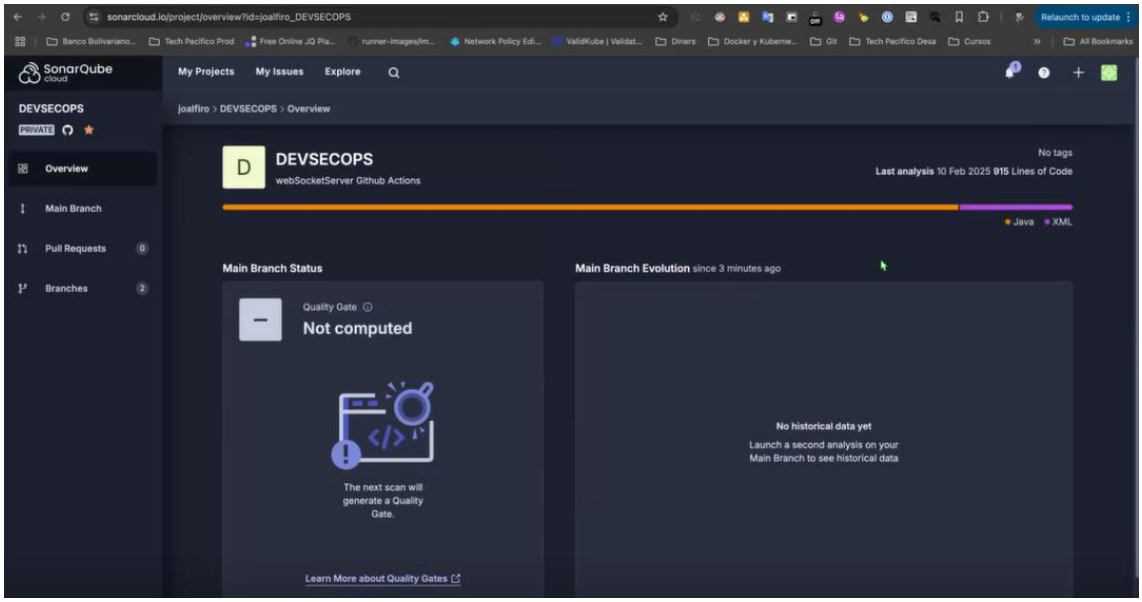


Figura 70. Vista General del Proyecto “DEVSECOPS” en SonarQube

En Main Branch en cambio, se verifica el análisis realizado que tiene 19 issues

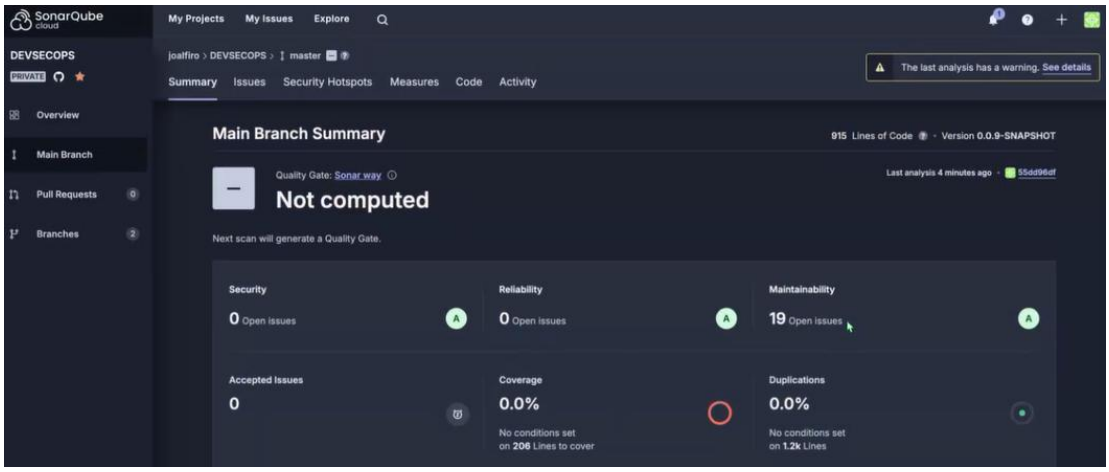


Figura 71. Resumen de la Rama Principal del Proyecto “DEVSECOPS” en SonarQube

Los permissions son necesarios para que el pipeline muestre resultados del análisis de Trivy

```yaml
.github > workflows > ! docker-build-deploy.yml
on:
  push:
    branches:

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    outputs:
      image_tags: ${{ steps.meta.outputs.tags }} # Output the tags from the metadata action
    permissions:      You, 4 days ago • Agregar GitHub Actions
      contents: read # Required for the Trivy scan
      packages: write   # Required for the Trivy scan
      security-events: write # Required for the Trivy scan
```

Figura 72. Configuración de Permisos en el Archivo docker-build-deploy.yml

La configuración de Trivy, es lo que hace el leer la fuente con análisis estático

```yaml
    - name: Run Trivy vulnerability scanner (Filesystem scan)      You, 4 days
      uses: aquasecurity/trivy-action@master # Use the Trivy action
      with:
        scan-type: 'fs'
        scan-ref: 'webSocketServer'
        format: 'table'
        severity: 'CRITICAL,HIGH'
        output: 'trivy-results.txt'
```

Figura 73. Configuración del Escaneo de Vulnerabilidades con Trivy en GitHub Actions

Posteriormente los resultados se guardan en un artifact por 7 días

```yaml
- name: Save Trivy scan results
  uses: actions/upload-artifact@v4 # Upload the Trivy scan results as an artifact
  with:
    name: trivy-scan-results
    path: trivy-results.txt
    retention-days: 7
```

Figura 74. Subida de los Resultados de Trivy como Artifact en GitHub Actions

Los resultados se pueden visualizar en Actions

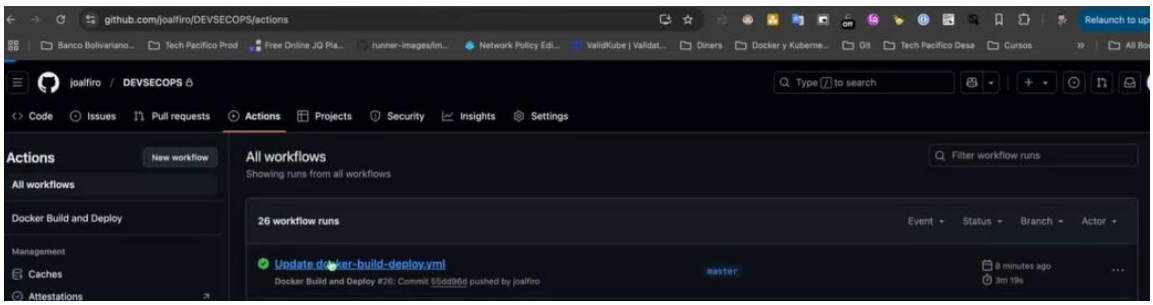


Figura 75. Historial de Ejecuciones del Workflow docker-build-deploy.yml en GitHub Actions

Sucesivamente, se guarda el análisis del escaneo de Trivy

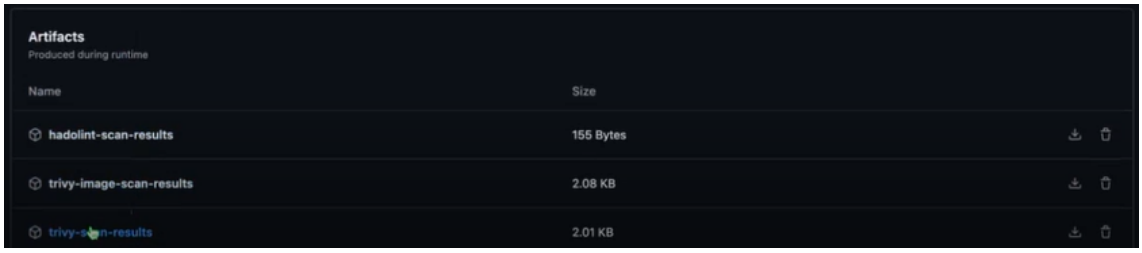


Figura 76. Artefactos de Escaneo Generados durante la Ejecución de GitHub Actions

Se procede a la descarga y obtención de las vulnerabilidades en el código fuente.

For OSS Maintainers: VEX Notice

If you're an OSS maintainer and Trivy has detected vulnerabilities in your project that you believe are not actually exploitable, consider issuing a VEX (Vulnerability Exploitability eXchange) statement. VEX allows you to communicate the actual status of vulnerabilities in your project, improving security transparency and reducing false positives for your users.
Learn more and start using VEX: https://aquasecurity.github.io/trivy/v0.57/docs/supply-chain/vex/repo#publishing-vex-documents

To disable this notice, set the TRIVY_DISABLE_VEX_NOTICE environment variable.

pom.xml (pom)
Total: 16 (HIGH: 15, CRITICAL: 1)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
ch.qos.logback:logback-classic ch.qos.logback:logback-core	CVE-2023-6378	HIGH	fixed	1.2.12	1.3.12, 1.4.12, 1.2.13	logback: serialization vulnerability in logback receiver https://avd.aquasec.com/nvd/cve-2023-6378
	CVE-2023-6481				1.4.14, 1.3.14, 1.2.13	logback: A serialization vulnerability in logback receiver https://avd.aquasec.com/nvd/cve-2023-6481
io.netty:netty-handler	CVE-2025-24970			4.1.92.Final	4.1.118.Final	SslHandler doesn't correctly validate packets which can lead to native crash when... https://avd.aquasec.com/nvd/cve-2025-24970
org.apache.tomcat.embed:tomcat-embed-core	CVE-2023-46589			9.0.75	11.0.0-M11, 10.1.16, 9.0.83, 8.5.96	tomcat: HTTP request smuggling via malformed trailer headers https://avd.aquasec.com/nvd/cve-2023-46589
	CVE-2024-34750				11.0.0-M21, 10.1.25, 9.0.90	tomcat: Improper Handling of Exceptional Conditions https://avd.aquasec.com/nvd/cve-2024-34750
	CVE-2024-50379				11.0.2, 10.1.34, 9.0.98	tomcat: RCE due to TOCTOU issue in JSP compilation https://avd.aquasec.com/nvd/cve-2024-50379
	CVE-2024-56337					tomcat: Incomplete fix for CVE-2024-50379 - RCE due to TOCTOU issue in... https://avd.aquasec.com/nvd/cve-2024-56337
org.springframework:spring-web	CVE-2016-1000027	CRITICAL		5.3.27	6.0.0	spring: HttpInvokerServiceExporter readRemoteInvocation method untrusted java deserialization https://avd.aquasec.com/nvd/cve-2016-1000027
	CVE-2024-22243	HIGH			6.1.4, 6.0.17, 5.3.32	springframework: URL Parsing with Host Validation https://avd.aquasec.com/nvd/cve-2024-22243
	CVE-2024-22259				6.1.5, 6.0.18, 5.3.33	springframework: URL Parsing with Host Validation https://avd.aquasec.com/nvd/cve-2024-22259
	CVE-2024-22262				5.3.34, 6.0.19, 6.1.6	springframework: URL Parsing with Host Validation https://avd.aquasec.com/nvd/cve-2024-22262
org.springframework:spring-webmvc	CVE-2024-38816				6.1.13	spring-webmvc: Path Traversal Vulnerability in Spring Applications Using RouterFunctions and FileSystemResource https://avd.aquasec.com/nvd/cve-2024-38816
	CVE-2024-38819				6.1.14	org.springframework:spring-webmvc: Path traversal vulnerability in functional web frameworks https://avd.aquasec.com/nvd/cve-2024-38819
org.yaml:snakeyaml	CVE-2022-1471			1.30	2.0	SnakeYaml: Constructor Deserialization Remote Code Execution https://avd.aquasec.com/nvd/cve-2022-1471
	CVE-2022-25857				1.31	snakeyaml: Denial of Service due to missing nested depth limitation for collections... https://avd.aquasec.com/nvd/cve-2022-25857

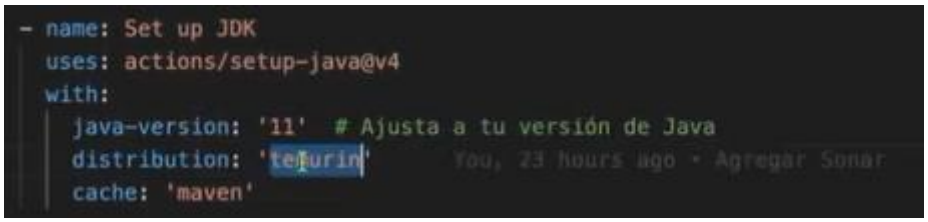
Figura 77. Reporte de Vulnerabilidades Generado por Trivy en Formato Tabular

El comando Checkout se utiliza para para descargarse el repositorio el Git Actions y esta descarga la fuente del último commit

```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v4 # Checkout the repository to the runner
    with:
      fetch-depth: 1 # Only fetch the latest commit
```

Figura 78. Paso de Checkout del Repositorio en GitHub Actions

La figura muestra la versión de java que se utiliza y una cache de Maven para la compilación



```yaml
- name: Set up JDK
  uses: actions/setup-java@v4
  with:
    java-version: '11'  # Ajusta a tu versión de Java
    distribution: 'temurin'
    cache: 'maven'
```

Figura 79. Configuración del Entorno Java (JDK) en GitHub Actions

En la figura se visualiza el step que almacena en la cache de sonar para que sea mucho más rápido

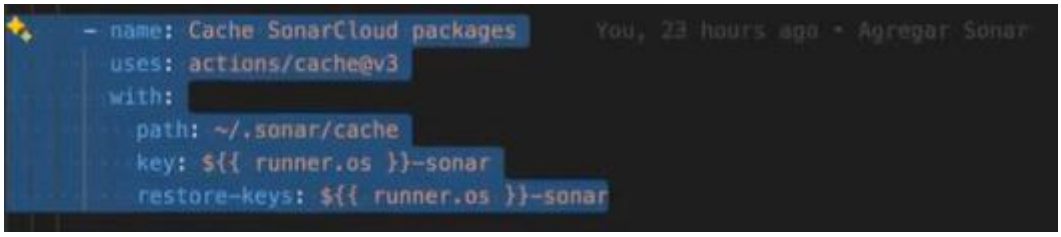


Figura 80. Almacenamiento en Caché de Paquetes SonarCloud en GitHub Actions

Este step almacena en la cache de maven dentro del mismo pipeline

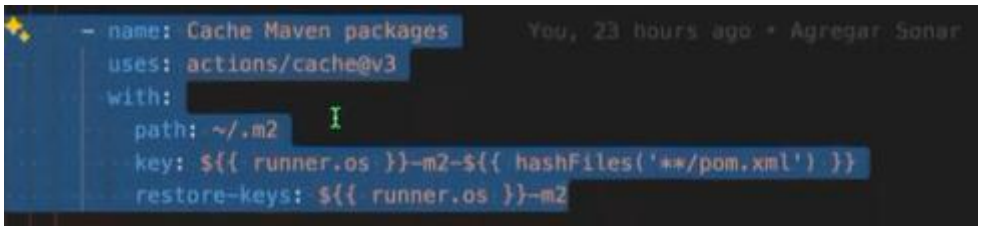


Figura 81. Almacenamiento en Caché de Paquetes Maven en GitHub Actions

Este comando indica la compilación de Maven

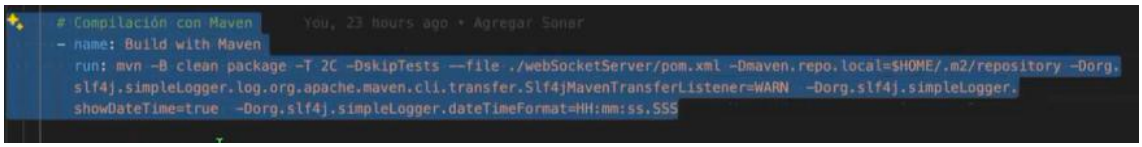


Figura 82. Ejecución de la Compilación con Maven en GitHub Actions

Se indica la compilación de sonar y el comando de maven para que ejecute sonar

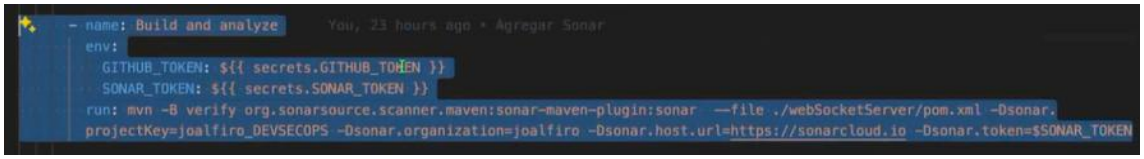


Figura 83. Paso "Build and Analyze" con SonarQube y Tokens de Autenticación en GitHub Actions

A continuación, configuración de pom.xml muestra la configuración y dependencias del proyecto

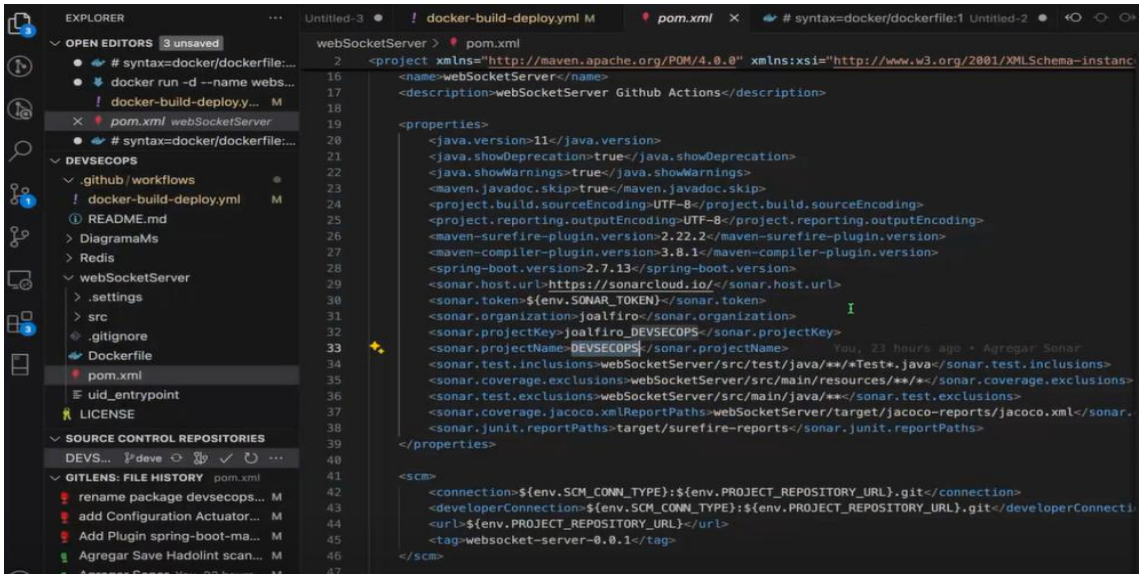


Figura 84. Configuración del Archivo pom.xml para Integración con SonarQube

Lo siguiente se realiza para verificar si el Dockerfile cumple con buenas prácticas

Figura 85. Configuración del Linter para Docker con Hadolint en GitHub Actions

Se verifica también en Dockerfile Linter – Hadolint

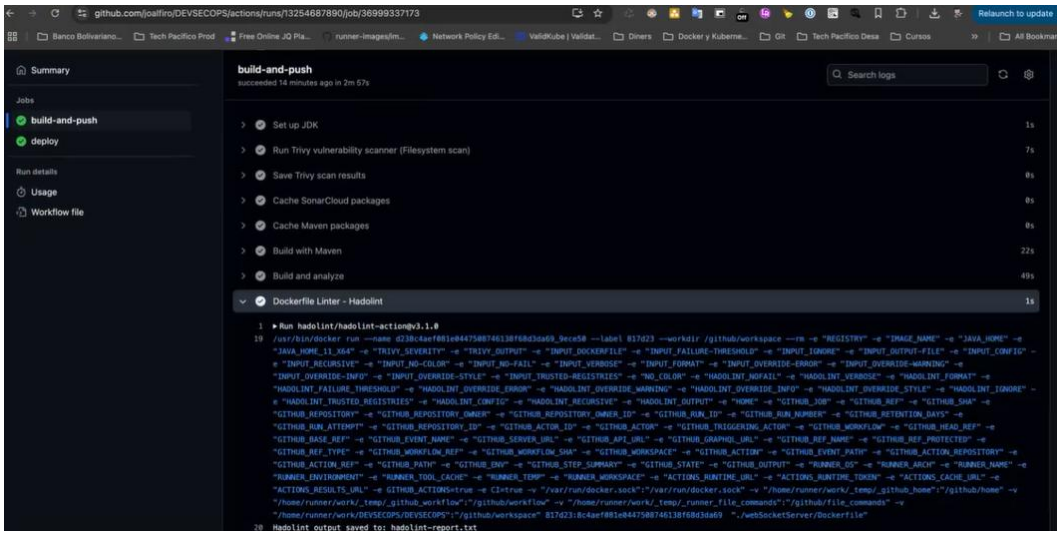


Figura 86. Salida del Linter Hadolint en GitHub Actions

Y también en Artifacts

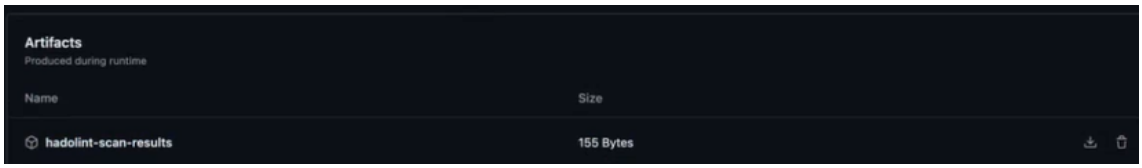
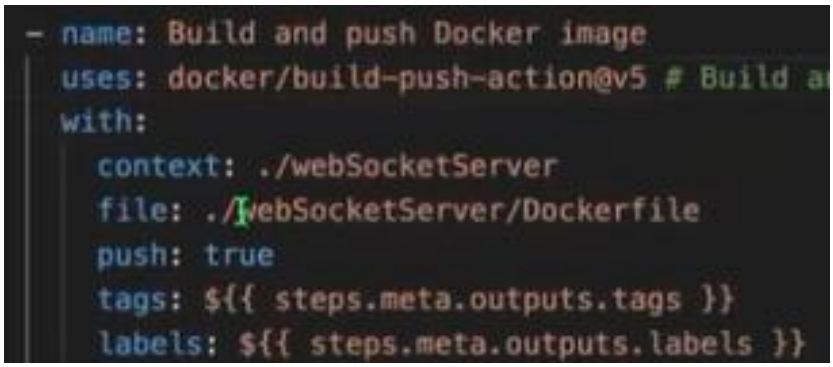


Figura 87. Artefacto “hadolint-scan-results” Generado durante la Ejecución de GitHub Actions

Y se obtiene la siguiente pantalla que no muestra error y por lo tanto, cumple con buenas prácticas

Figura 88. Vista del Archivo hadolint-report.txt con Resultados del Linter Docker

Se crea y se guarda la imagen de Docker para compilar la imagen de la aplicación por medio del Dockerfile



. Figura 89. Paso "Build and Push Docker Image" en GitHub Actions

Visualizamos a continuación, Deploy en el que se configuraron las credenciales

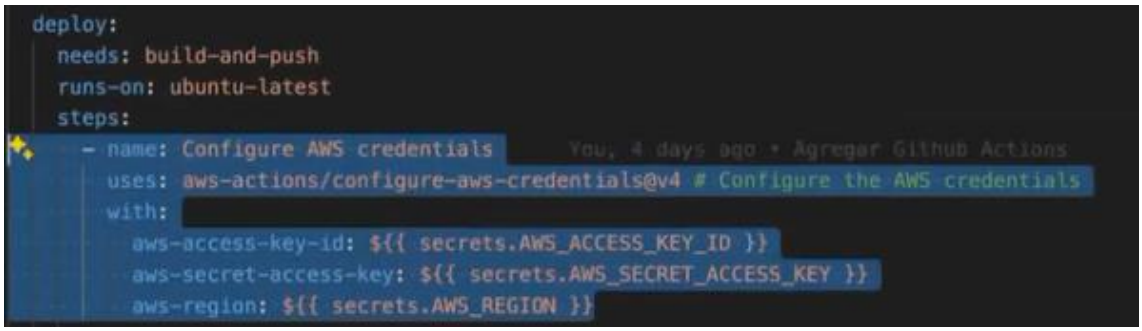


Figura 90. Configuración de Credenciales de AWS en GitHub Actions

Para despliegue, se loguea con el repositorio de GitHub, se descarga la imagen y se detiene el contenedor anterior y se levanta la aplicación

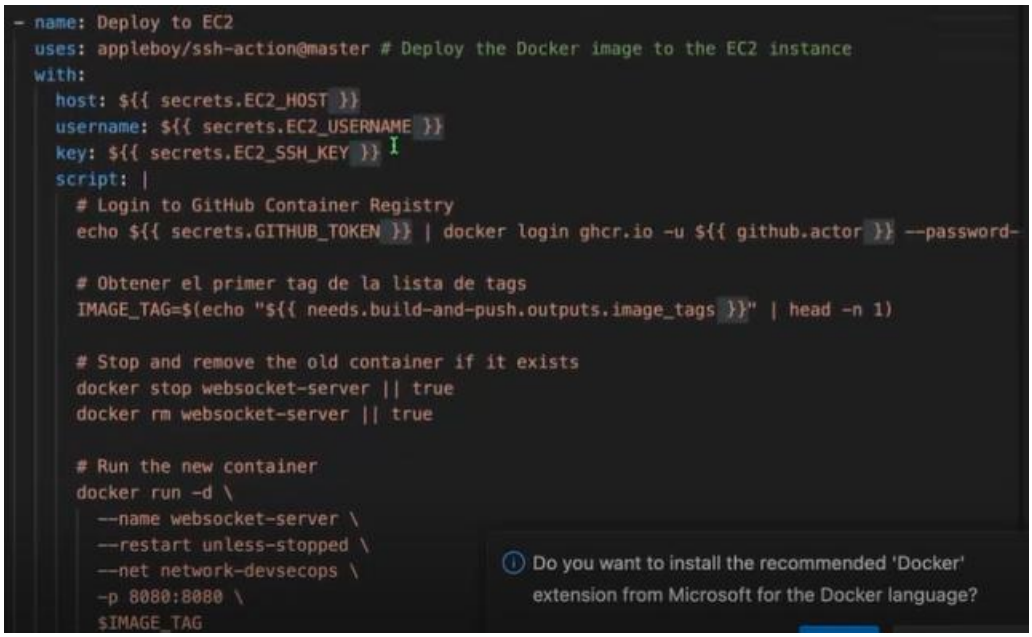


Figura 91. Paso de Despliegue de la Imagen Docker en la Instancia EC2 desde GitHub Actions

De igual manera de selecciona en Connect

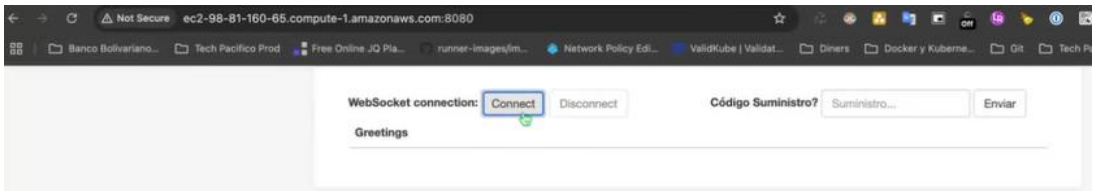


Figura 92. Interfaz de Conexión WebSocket en la Instancia EC2

Se verifica la aplicación levantada, el contenedor y la imagen

Figura 93. Visualización de Contenedores Docker en la Instancia EC2 con WebSocket Server y Redis

Se configura docker-compose.yml para instalar redis EC2 de la siguiente manera:

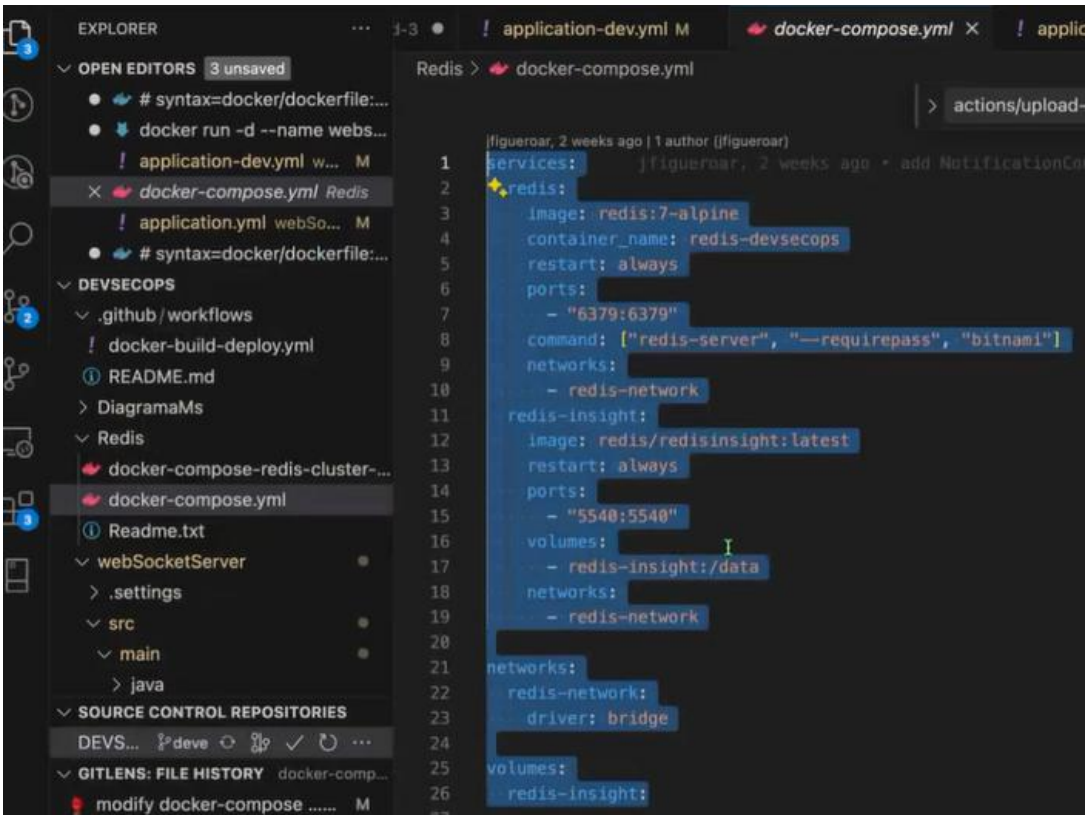


Figura 94. Contenido de docker-compose.yml para Redis y WebSocket-Servero3-mini

4.1.3.4. Configuración de redis

Para esto se configura la regla en AWS para que se conecta con la base de datos en memoria redis

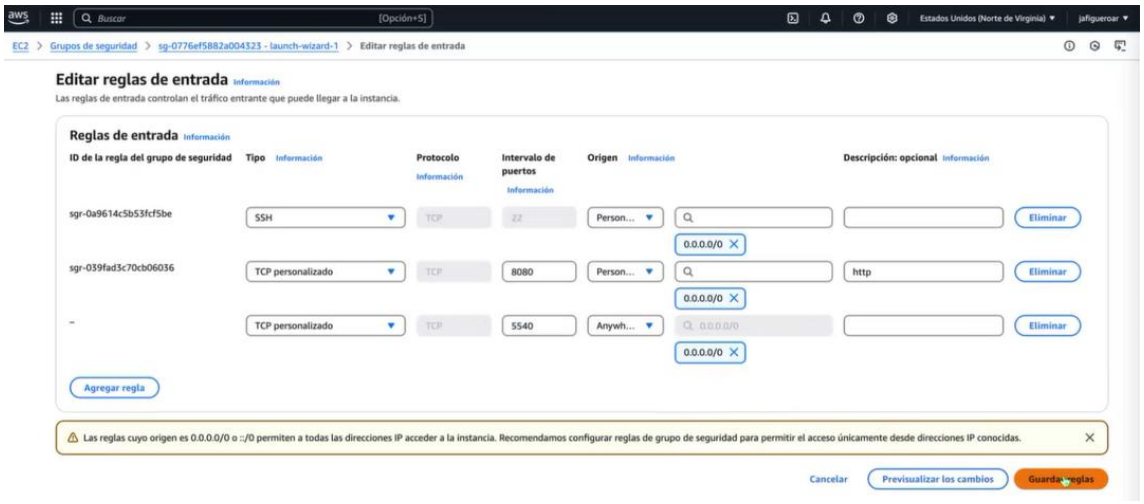


Figura 95. Edición de Reglas de Entrada en el Grupo de Seguridad de AWS

Seguidamente se verifica la conexión de redis

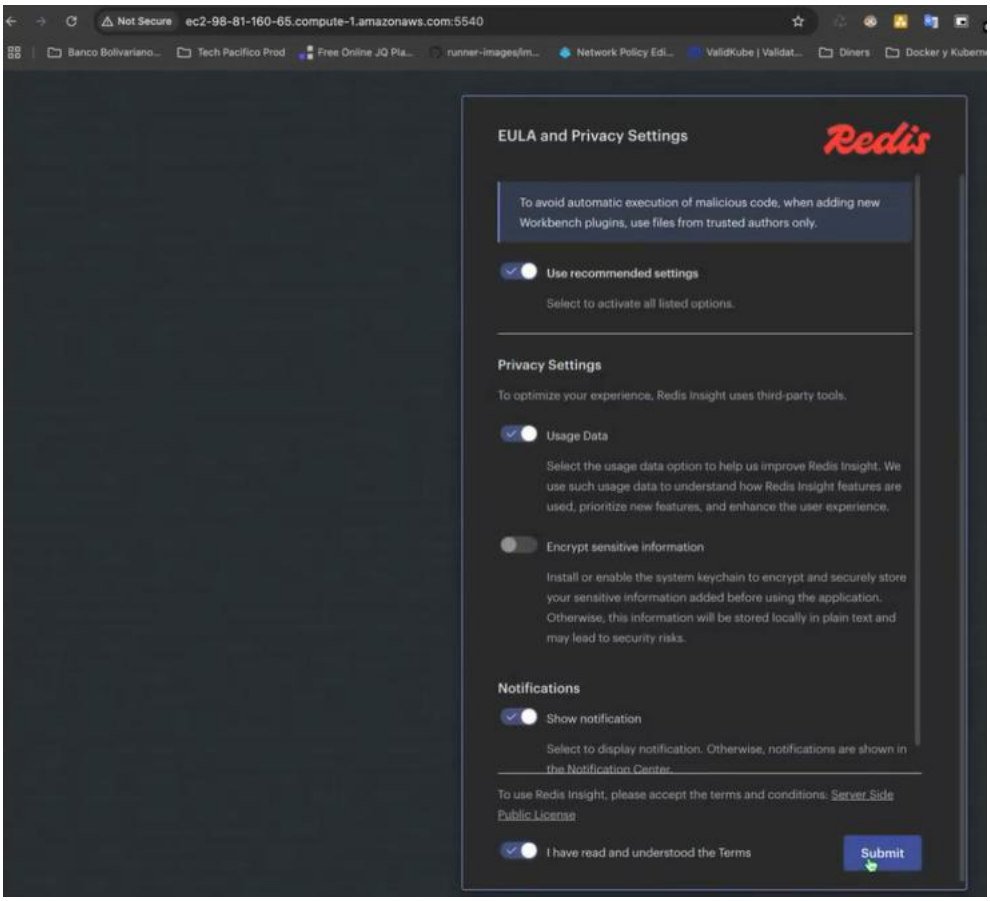


Figura 96. EULA y Configuración de Privacidad en la Interfaz de Redis

Y se configura redis de la siguiente manera:

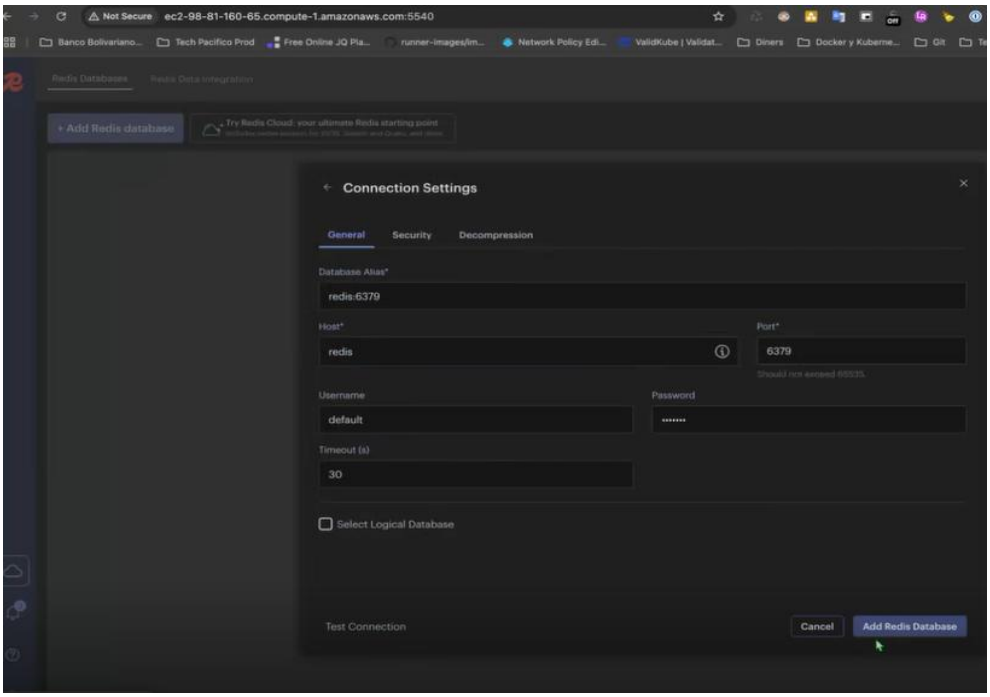


Figura 97. Configuración de Conexión a la Base de Datos Redis en la Interfaz Web

A continuación, se verifica que se encuentre funcionando Redis

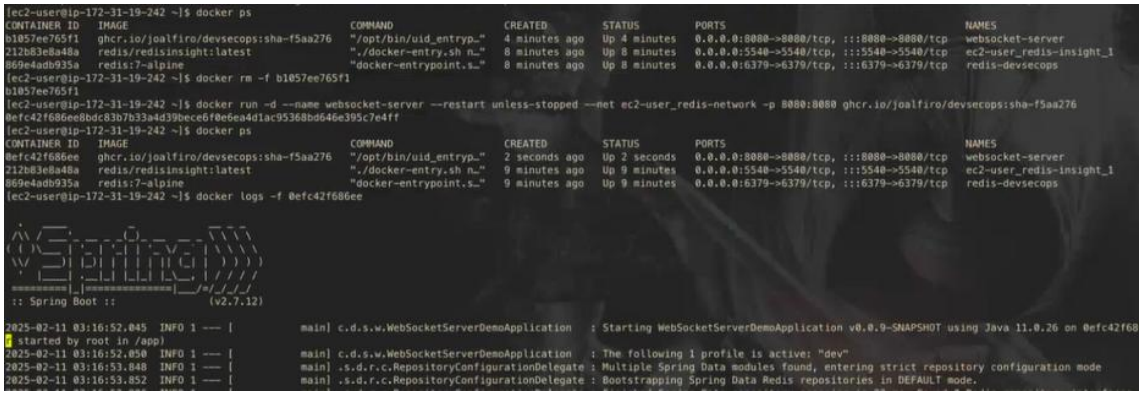


Figura 98. Ejecución de Contenedores Docker y Registro de Logs de la Aplicación Spring Boot

De esta manera se verifica que se encuentre funcionando la aplicación y consumiendo el microservicio

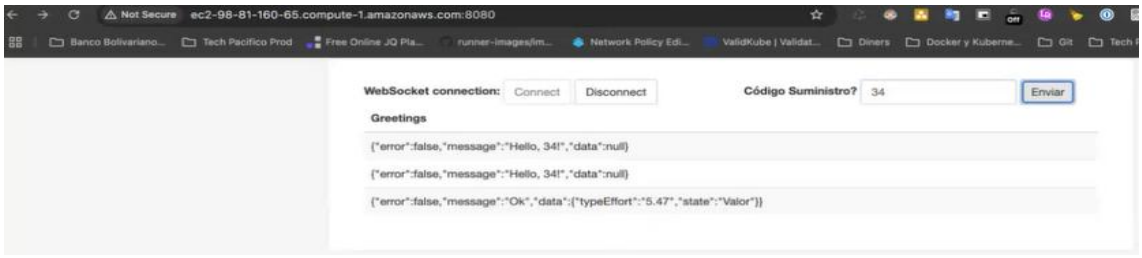


Figura 99. Interfaz de Prueba para WebSocket con Mensajes Enviados y Recibidos

Para poder consumir el webhook que expone el microservicio se puede utilizar el siguiente ejemplo: #curl Cliente Postman

curl --location
'http://localhost:8081/v1/notification/sessionId/23456/webhook/resultRisk'

--header 'Content-Type: application/json'

--data '{

"dinBody": { "state": "Valor", "typeEffort": "6.47" } }'

Para enviar una respuesta al websocket de un mensaje con la estructura definida en el dinbody

Y entonces se procede a verificar redis

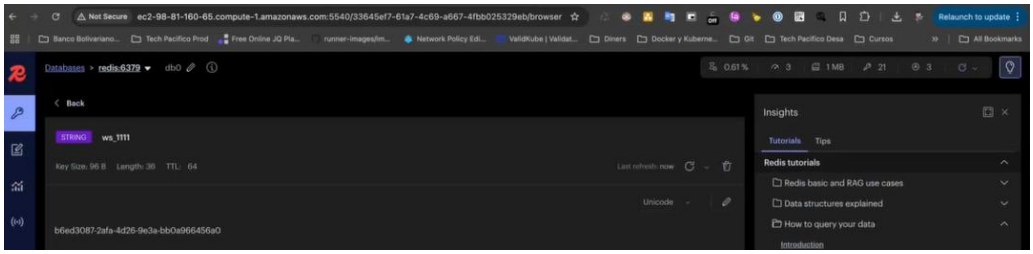


Figura 100. Visualización de una Clave en RedisInsight

Y configurar Dockerfile para compilar la aplicación con maven

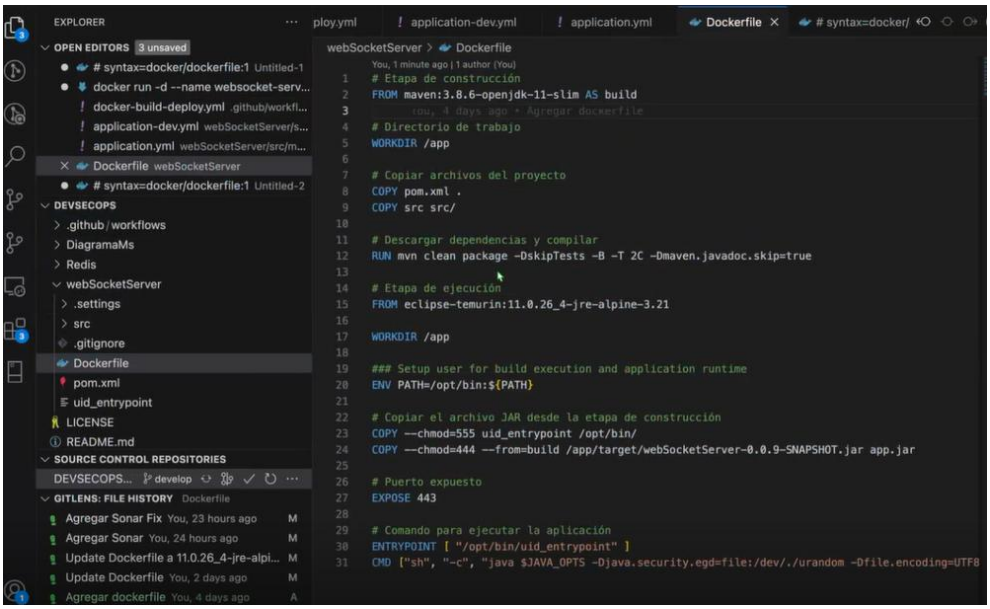


Figura 101. Contenido del Dockerfile para la Aplicación WebSocketServer

Se configura uid_entrypoint para saber si la aplicación es vulnerable para usuarios y password dentro del sistema operativo

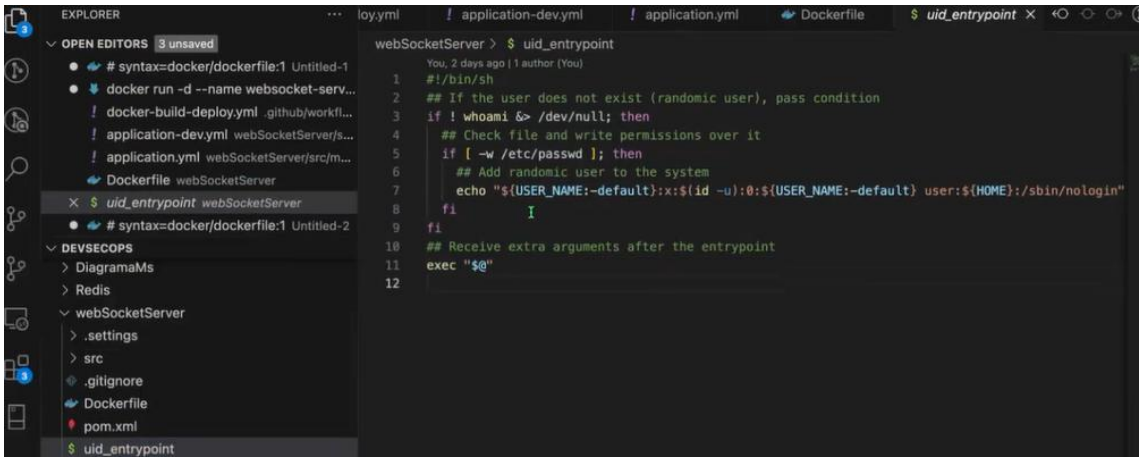


Figura 102. Script de Entrypoint para la Aplicación WebSocketServer

4.1.3.5. Configuración de GitHub dashboard.

Es el panel de control principal de GitHub donde se puede visualizar la información sobre el repositorio, el proyecto y la actividad en la plataforma.

Para ello se selecciona el repositorio

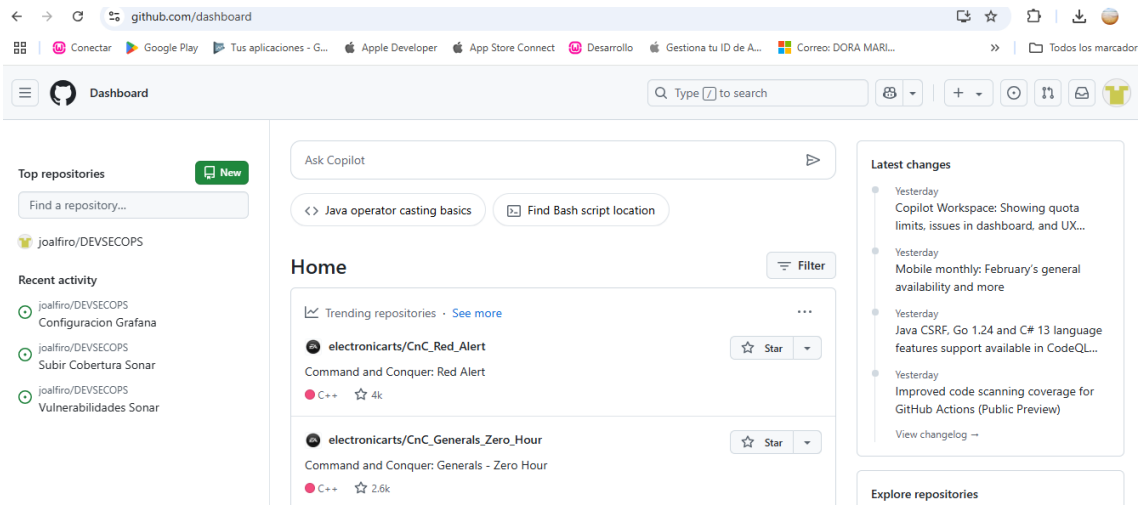


Figura 103. Vista del Panel de Control (Dashboard) en GitHub

En Code se muestra el branch: master, develop

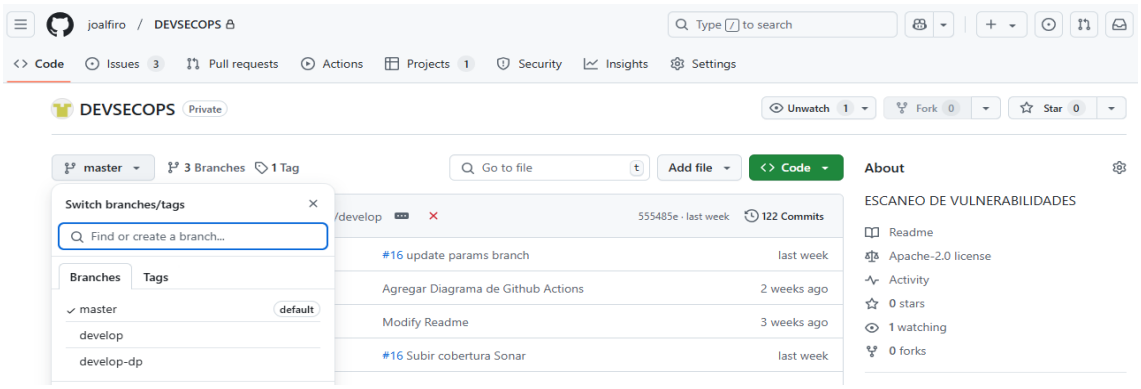


Figura 104. Selección de Ramas y Tags en el Repositorio "DEVSECOPS" en GitHub

En Issues se pueden visualizar la configuración grafana, las vulnerabilidades de sonar y subir la cobertura de sonar

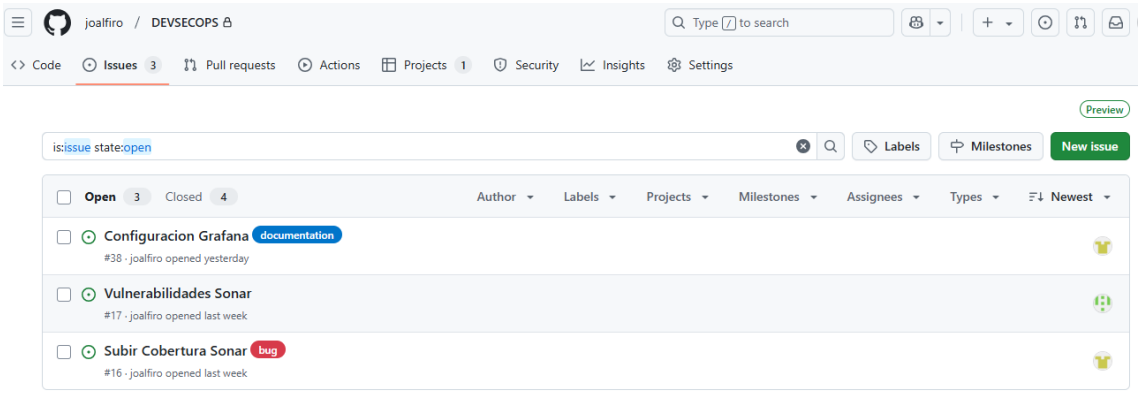


Figura 105. Listado de Issues Abiertos en el Repositorio "DEVSECOPS" en GitHub

En Pull requests GitHub se muestra si el PR está pendiente, aprobado o con conflictos

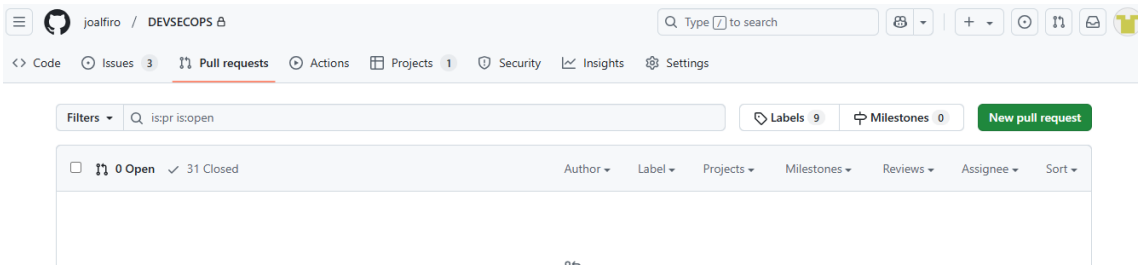


Figura 106. Sección de Pull Requests en el Repositorio "DEVSECOPS" en GitHub

El comando Actions en cambio, indica el estado de las ejecuciones de los workflows

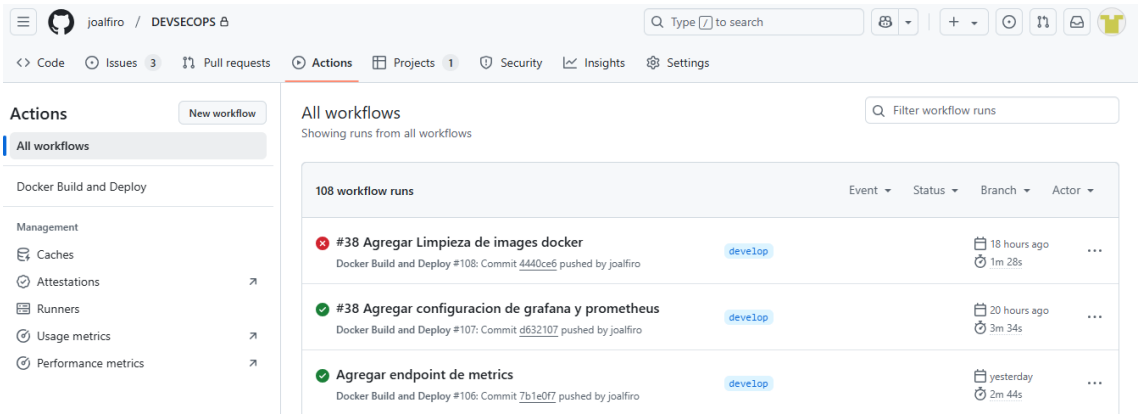


Figura 107. Historial de Workflows en el Repositorio "DEVSECOPS" de GitHub Actions

Podemos visualizar también Projects donde se indica el proyecto de la creación de un microservicio para el pago de servicios

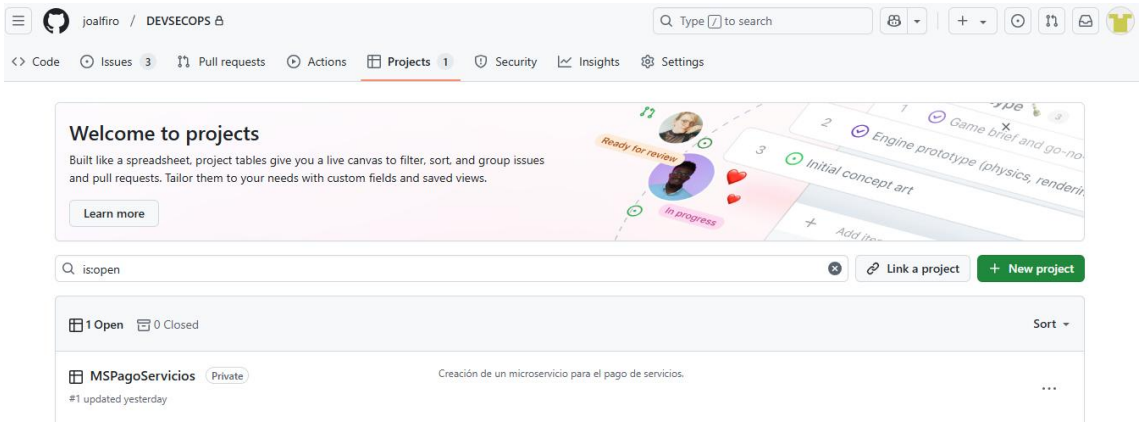


Figura 108. Sección de Proyectos en el Repositorio "DEVSECOPS" en GitHub

4.1.3.6. Configuración de Grafana

Para la configuración de Grafana, dentro de la consola de AWS se selecciona EC2

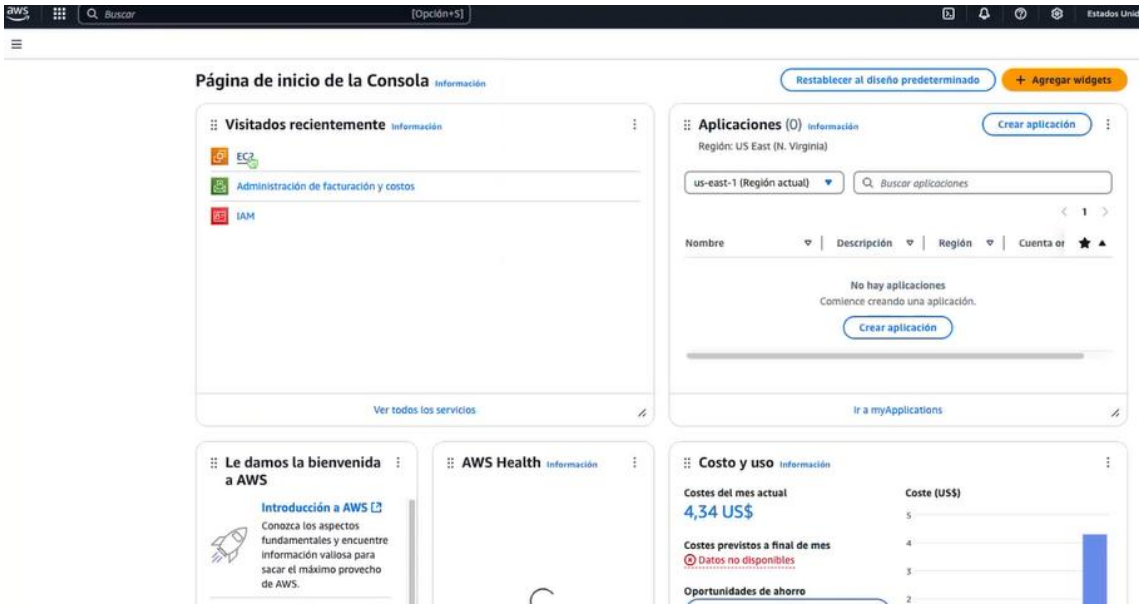


Figura 109. Vista de la Consola de AWS con Información de Costos y Aplicaciones

Después se selecciona la instancia y se realiza clic en Conectar

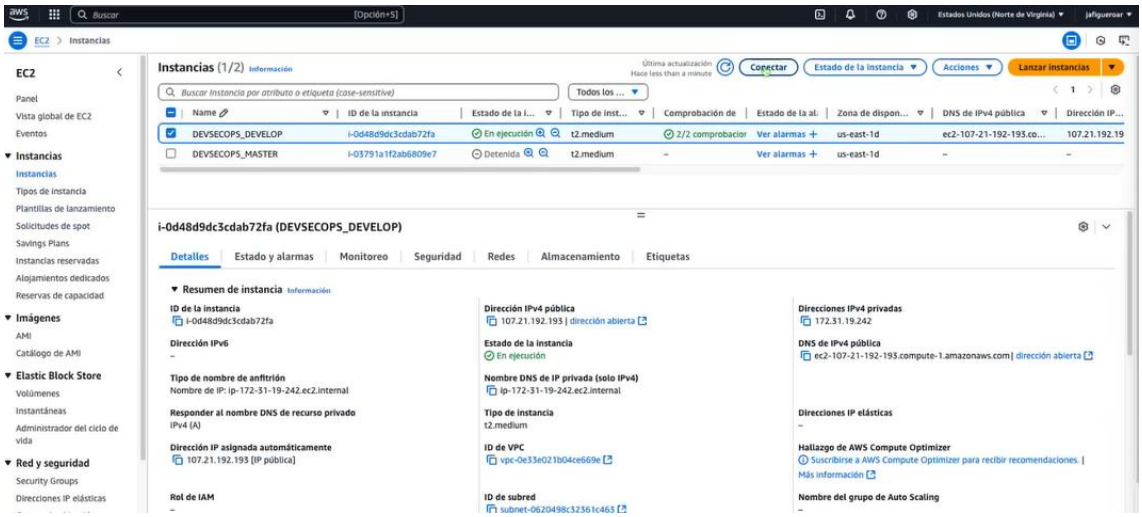


Figura 110. Visualización de Instancias EC2 “DEVSECOPS_MASTER” y “DEVSECOPS_DEVELOP” en AWS

De esta manera se visualiza al realizar clic en Conectar

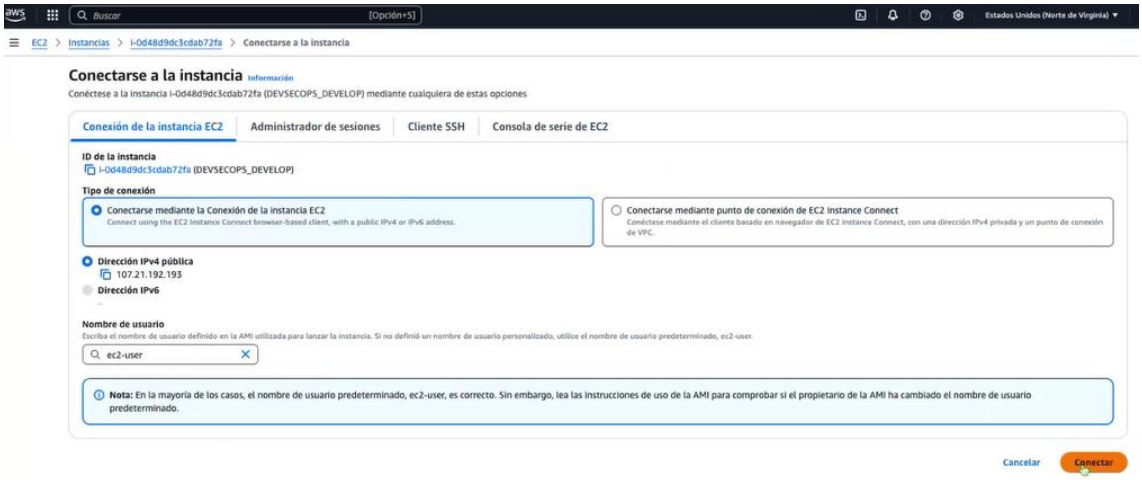


Figura 111. Opciones de Conexión para la Instancia “DEVSECOPS_DEVELOP” en AWS

Y lo que sucede con ello es que se conecta el browser a la máquina

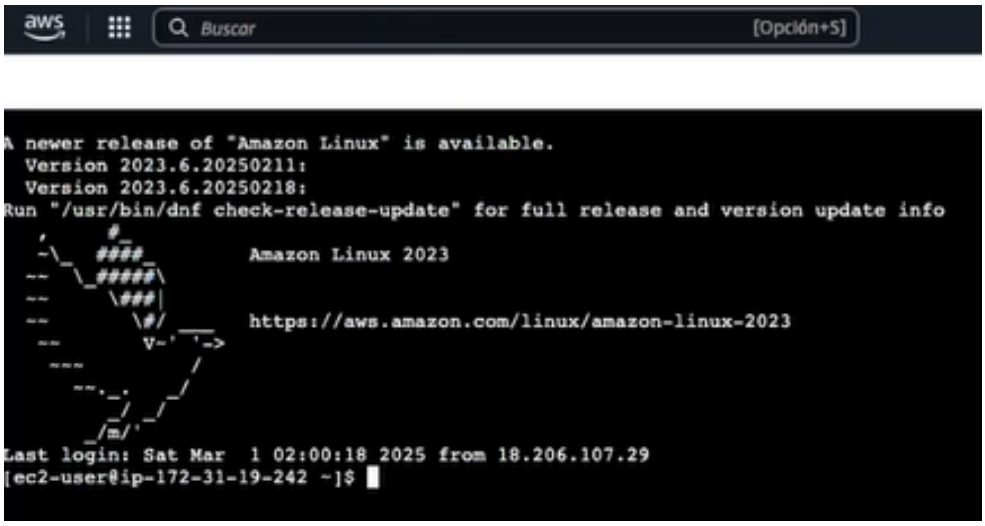


Figura 112. Sesión de la Instancia EC2 con Amazon Linux 2023 a través de la Consola

Se visualiza todo lo que se está ejecutando que en este caso serían grafana y prometeus es un recolector de métricas

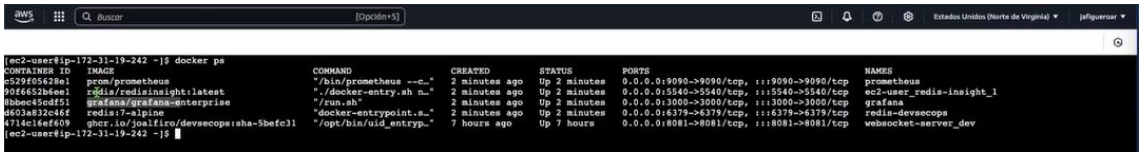


Figura 113. Listado de Contenedores en la Instancia EC2 con Amazon Linux 2023

A continuación se configura como servicio Docker-compose.yaml lo que hace es agrupar todos los contenedores y servicios para que se levante

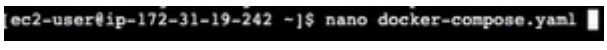


Figura 114. Edición del Archivo docker-compose.yml con Nano en la Instancia EC2

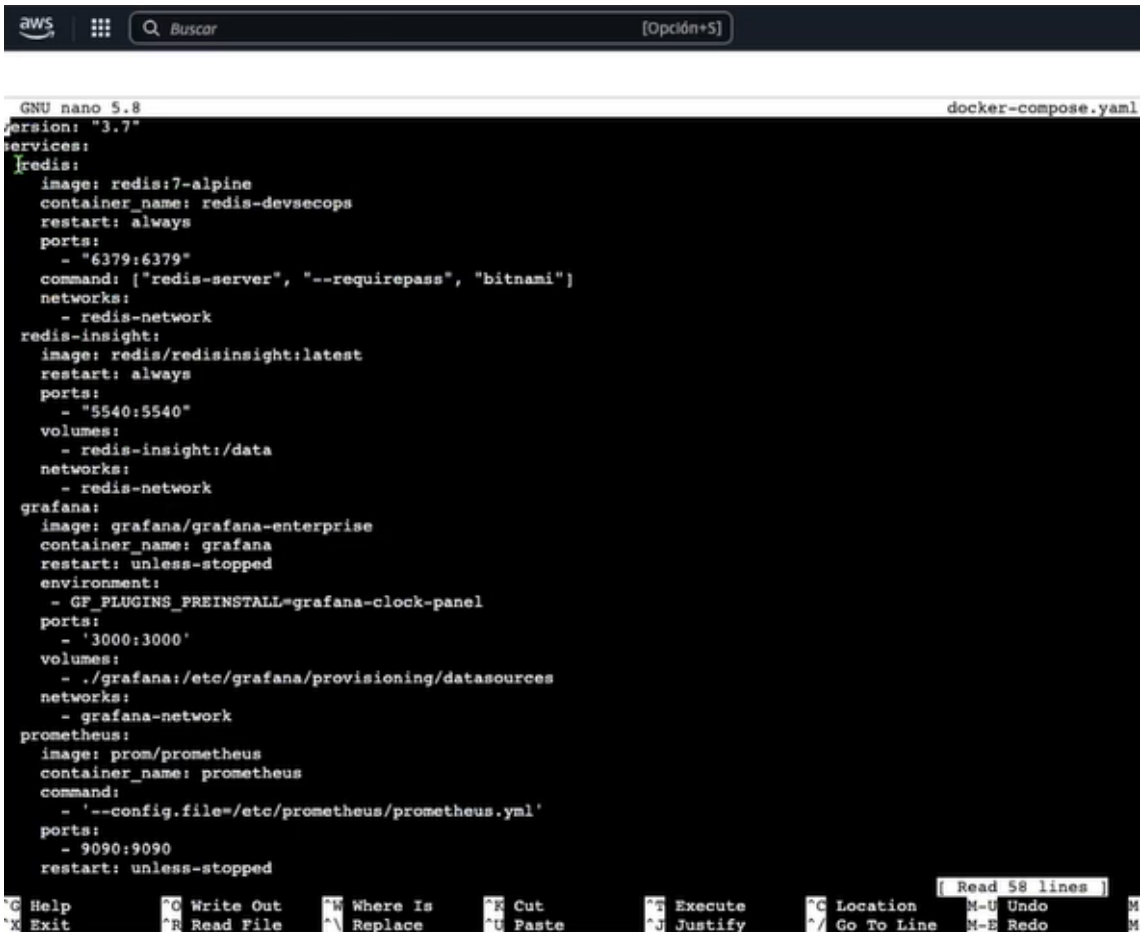


Figura 115. Contenido de docker-compose.yml para Redis, RedisInsight, Grafana y Prometheus en la Instancia EC2

La configuración de grafana se encuentra expuesto en el puerto 3000

Figura 116. Configuración del Contenedor Grafana en el Archivo docker-compose.yml

Y se visualiza lo que contiene Grafana internamente

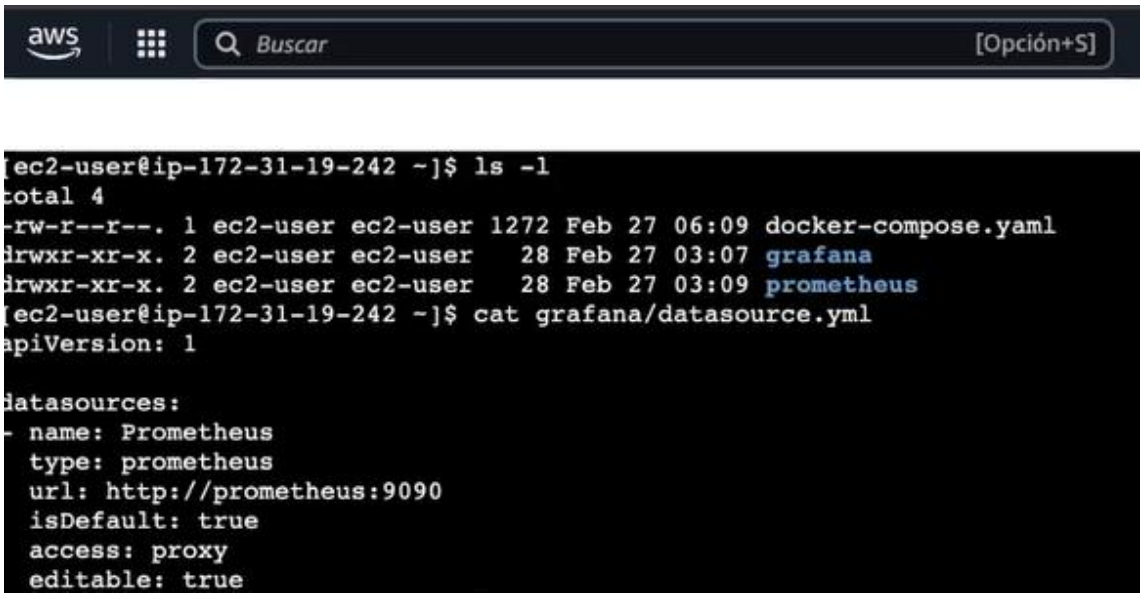


Figura 117. Estructura de Archivos en la Instancia EC2 y Configuración de datasource.yml para Grafana

Se adjunta también la configuración de prometheus que se encuentra expuesto en el puerto 9090

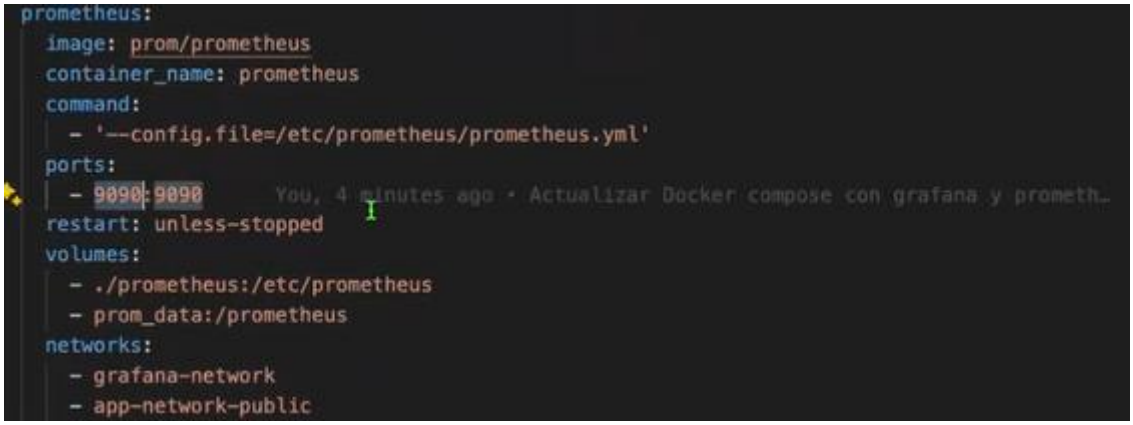


Figura 118. Configuración del Contenedor Prometheus en docker-compose.yml

Y esto es lo que contiene prometheus internamente

```
[ec2-user@ip-172-31-19-242 ~]$ cat prometheus/prometheus.yml
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
alerting:
  alertmanagers:
  - static_configs:
    - targets: []
    scheme: http
    timeout: 10s
    api_version: v2
scrape_configs:
- job_name: prometheus
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /actuator/prometheus
  scheme: http
  static_configs:
  - targets:
    - websocket-server_dev:8081
```

Figura 119. Configuración de Prometheus Apuntando al WebSocket-Server en el Puerto 8081

A continuación, se ingresa a Grafana en http://localhost:3000 usuario: admin, contraseña: admin

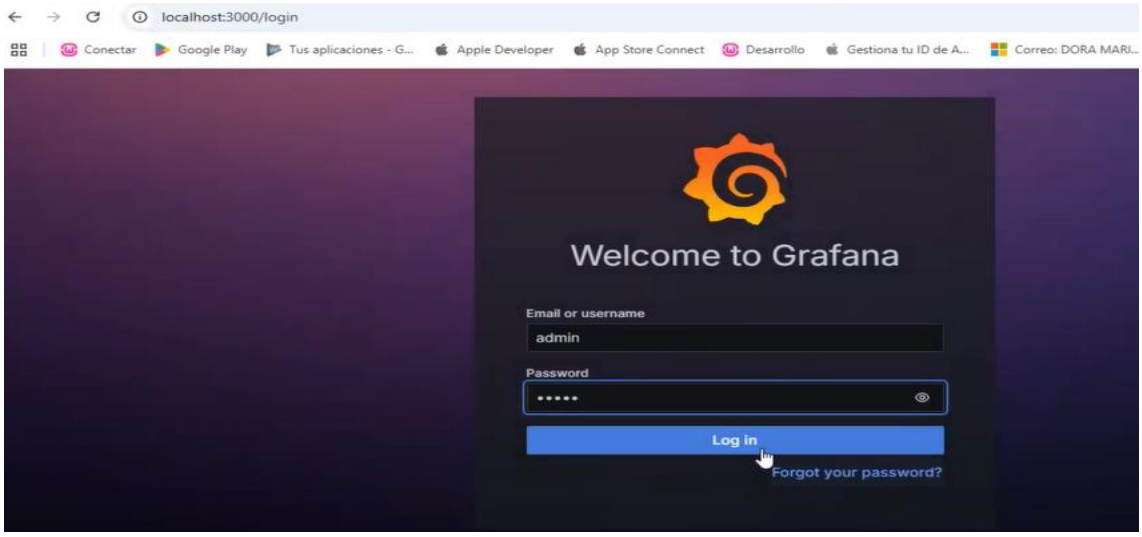


Figura 120. Pantalla de Inicio de Sesión en Grafana

Y se de esta manera se puede visualizar una vez que se ha ingresado a Grafana

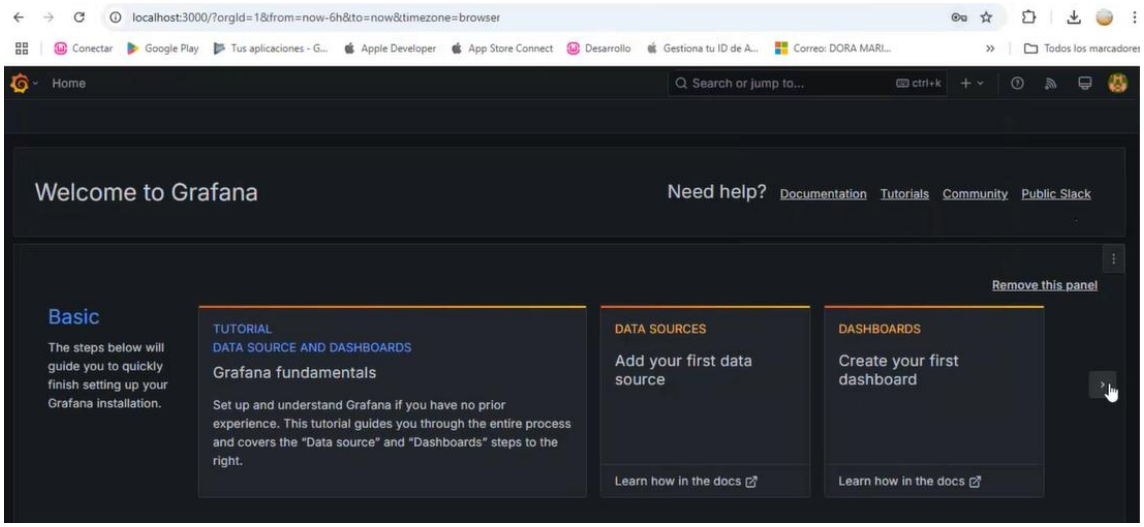


Figura 121. Panel de Bienvenida en la Interfaz Principal de Grafana

Una vez dentro, se continúa configurando y para esto se ingresa en home/dashboards

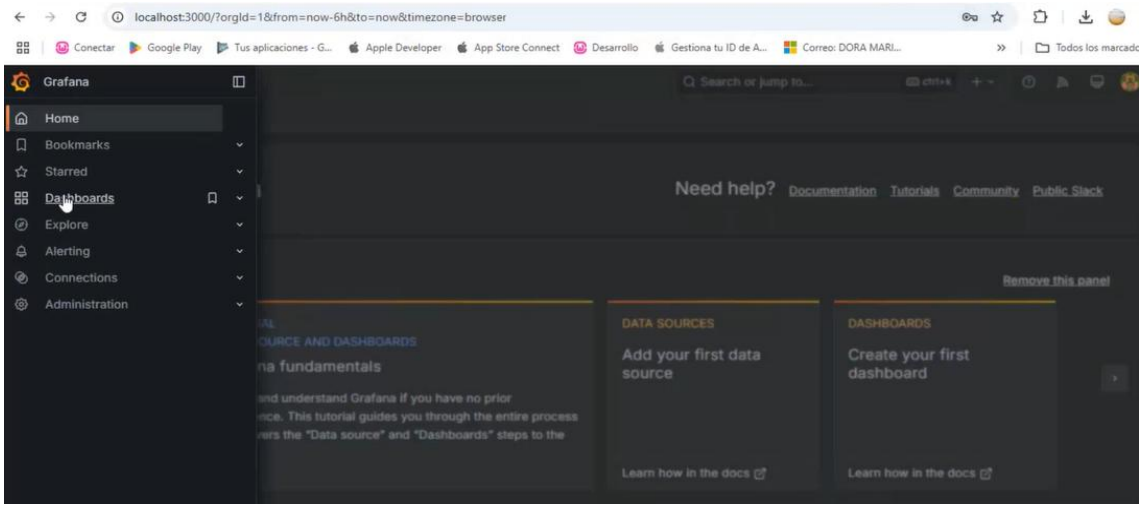


Figura 122. Menú Lateral de Grafana con la Sección de Dashboards

Y se selecciona New/Import

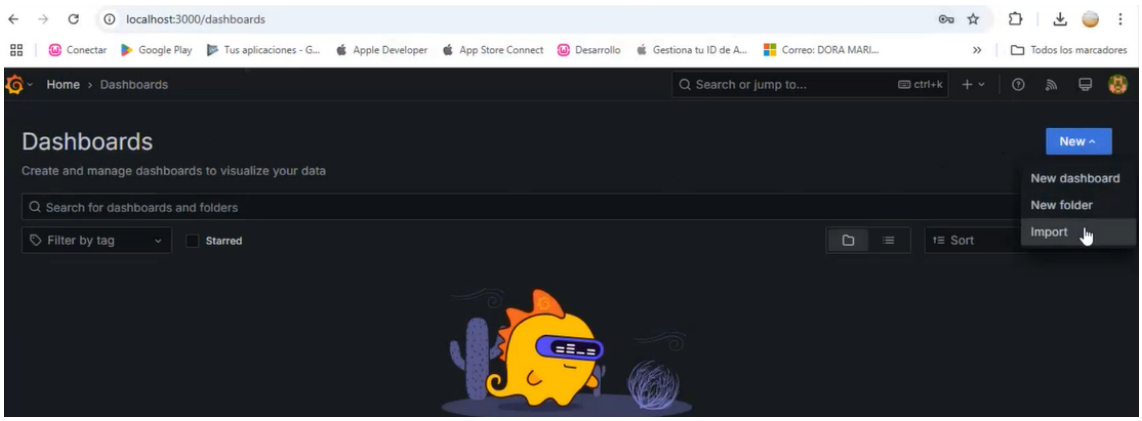


Figura 123. Creación e Importación de Dashboards en Grafana

Se ingresamos el id para importar el dashboard y clic en load

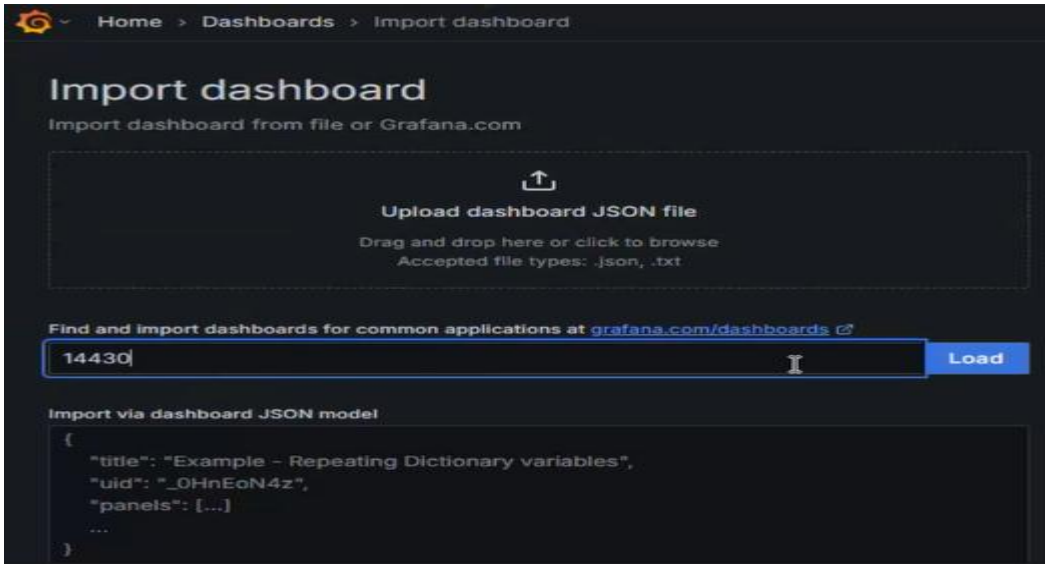


Figura 124. Importación de un Dashboard por ID en Grafana

A continuación, se selecciona configure a new data source

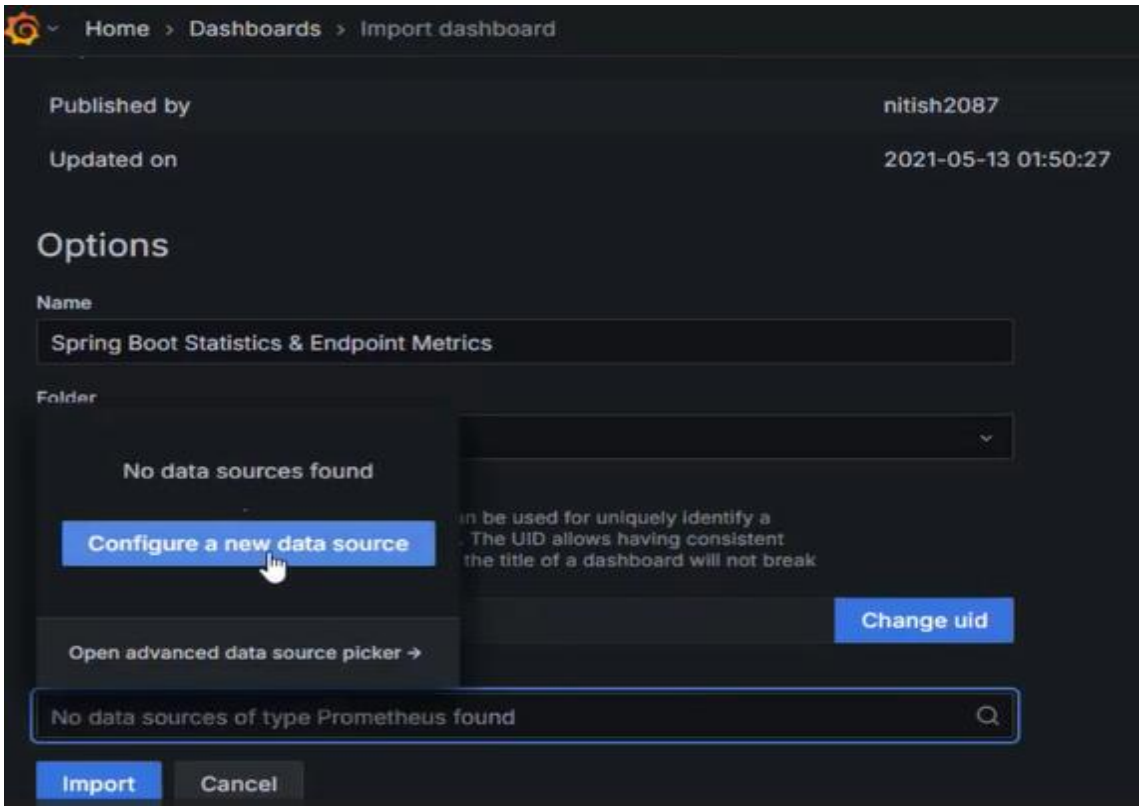


Figura 125. Configuración de una Nueva Fuente de Datos para el Dashboard en Grafana

Y se selecciona Prometheus

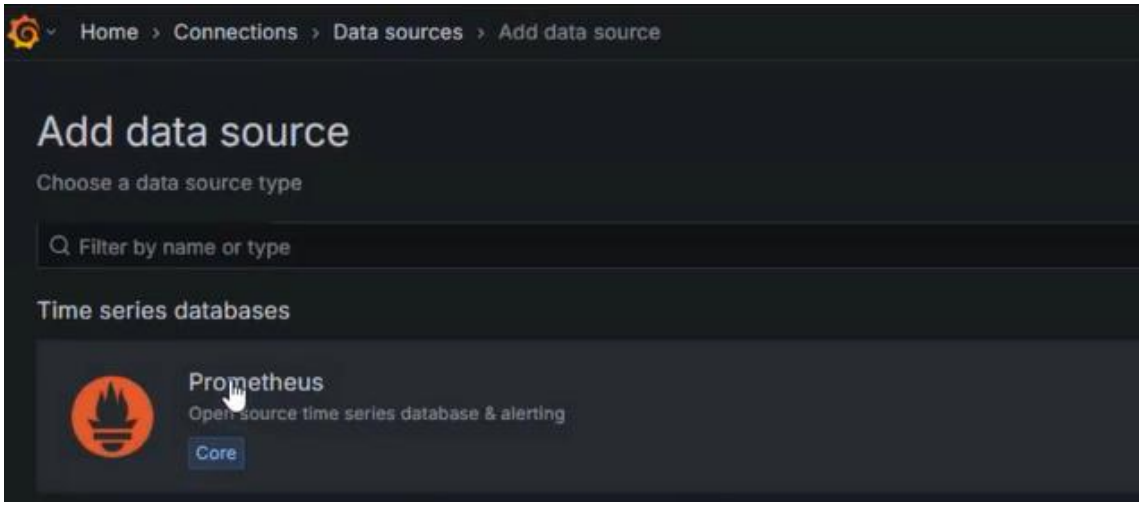


Figura 126. Selección de Prometheus como Nueva Fuente de Datos en Grafana

Se ingresa la url http://localhost:9090 y clic en Save & test

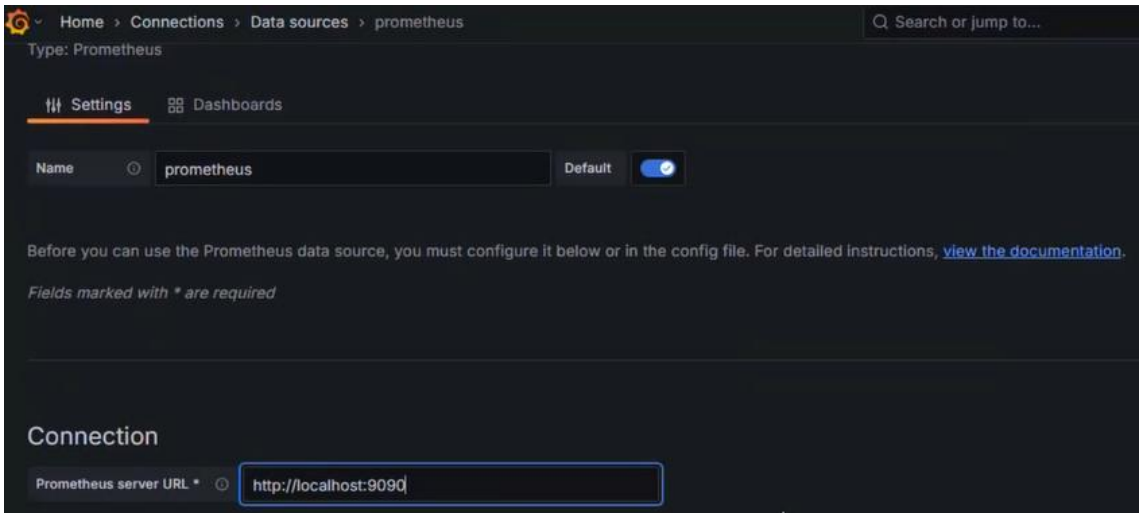


Figura 127. Configuración de la Fuente de Datos Prometheus en Grafana

Se selecciona prometheus y clic en Import

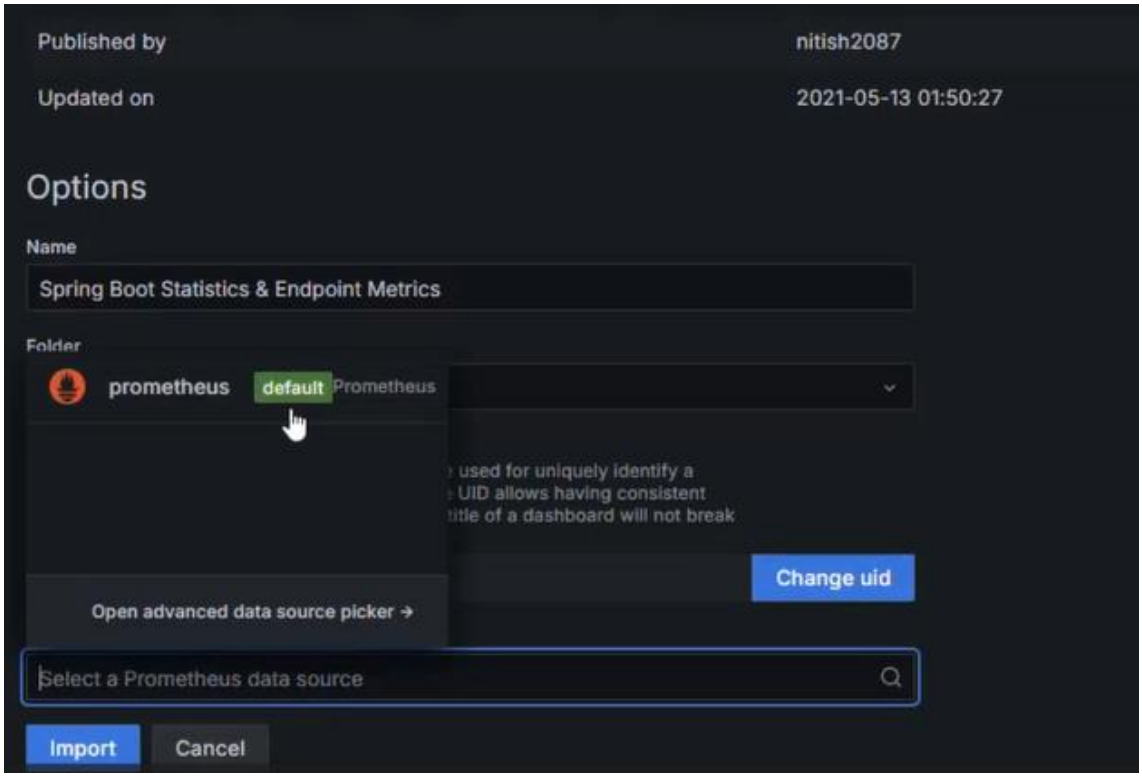


Figura 128. Importación del Dashboard “Spring Boot Statistics & Endpoint Metrics” con Fuente de Datos Prometheus en Grafana

Y es así como se indica:

- Uptime: indica el tiempo en segundos desde que la aplicación se inició.

- Heap Used: muestra la cantidad de memoria utilizada en el heap de la JVM.
- Non-Heap Used: memoria utilizada fuera del heap
- Start time: resta el tiempo actual (time ()) menos el tiempo de inicio (process_start_time_seconds) para obtener la cantidad de segundos en ejecución.
- CPU Usage: Muestra el uso de CPU en segundos por unidad de tiempo.
- Load Average: Muestra la carga promedio del sistema en los últimos 1, 5 o 15 minutos.

Figura 129. Panel de Monitoreo de Spring Boot y Endpoint Metrics en Grafana

La figura a continuación muestra:

HTTP Server Requests Count: Cuenta el número total de solicitudes HTTP recibidas

HTTP Server Requests Sum: para obtener la suma total de solicitudes HTTP recibidas en un servidor web.

HTTP Server Requests Max: para obtener el máximo número de solicitudes HTTP en un periodo de tiempo en Prometheus.

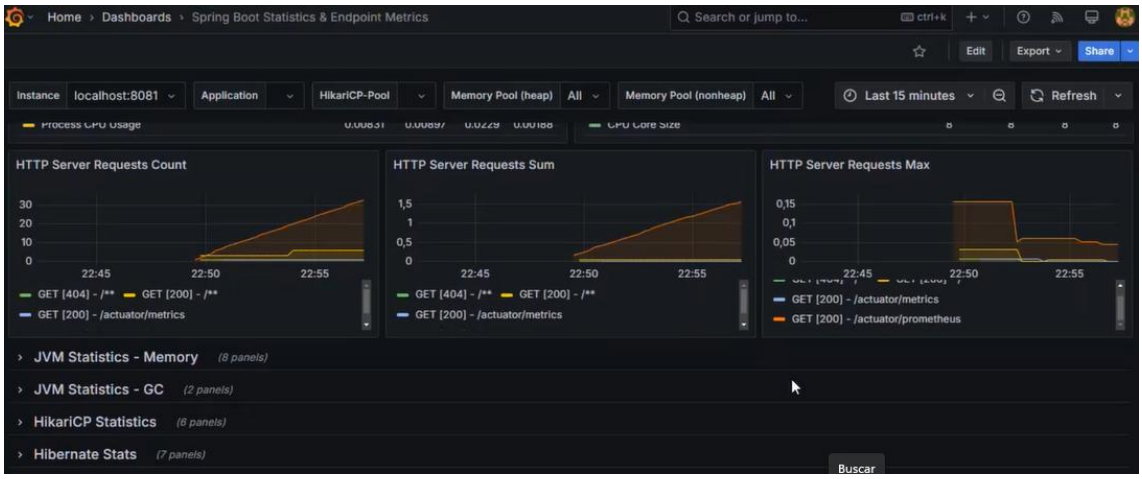


Figura 130. Visualización de HTTP Server Requests y Métricas de Endpoints en Grafana

4.1.3.7. Configuración para escaneo de vulnerabilidades en la imagen Docker

La imagen está compuesta ghcr (GitHub conteiner register), este register está alojando en el mismo Git Hub, GitHub Action al igual que AWS tiene su propio register pero en este caso es gratis, al momento de compilar la imagen con Docker este tiene que subir a un sitio para que pueda ser alojada, necesitamos publicarla para que AWS la pueda ver.

Por ejemplo, si se compila el Docker en la máquina, se tiene las imágenes, pero estas imágenes no van a ser públicas van a ser privadas porque van a estar alojadas en la máquina.

```
[ec2-user@ip-172-31-19-242 ~]$ docker images
REPOSITORY                    TAG          IMAGE ID       CREATED             SIZE
ghcr.io/joalfiro/devsecops    sha-ca46b66  14bde86f7f8b   36 minutes ago      201MB
ghcr.io/joalfiro/devsecops    sha-091e9a7  1b7a75bb496f   About an hour ago   201MB
ghcr.io/joalfiro/devsecops    sha-b621f1d  1500db51af59   22 hours ago        169MB
ghcr.io/joalfiro/devsecops    sha-18227ea  814bf634e547   22 hours ago        169MB
ghcr.io/joalfiro/devsecops    sha-929dcf3  db35ca19e909   22 hours ago        169MB
ghcr.io/joalfiro/devsecops    sha-fd87aa9  44ed893bbe49   22 hours ago        169MB
ghcr.io/joalfiro/devsecops    sha-38e6187  afa1112104df   23 hours ago        169MB
```

Figura 131. Listado de Imágenes Docker Etiquetadas con SHA en la Instancia EC2

Para que se puedan compartir entre personas o entre máquinas tiene que ser alojada en un register, en este caso se ha utilizado AWS, AZURE, GOOGLE. Las otras cuentas en la nube son de paga y se generan rubros económicos por tener ese register alojado, pero GitHub tiene la opción de tenerlo alojado gratis hasta 10 GB.

Cuando ghcr (GitHub conteiner register) se compila, ejecuta, se almacena en una sección que se llama package pero siempre se lo encontrará a nivel de usuario como muestra la imagen en joalfiro/Packages/devsecops (ghcr (GitHub conteiner register))

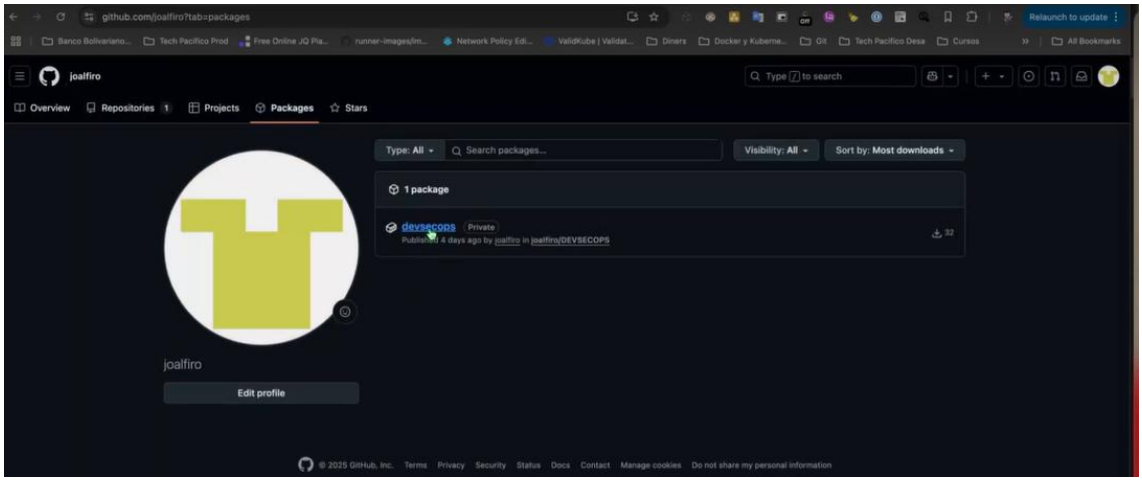


Figura 132. Vista de Paquetes en el Perfil de GitHub

Para lo cual se abre y se puede visualizar como se muestran todas las imágenes almacenadas

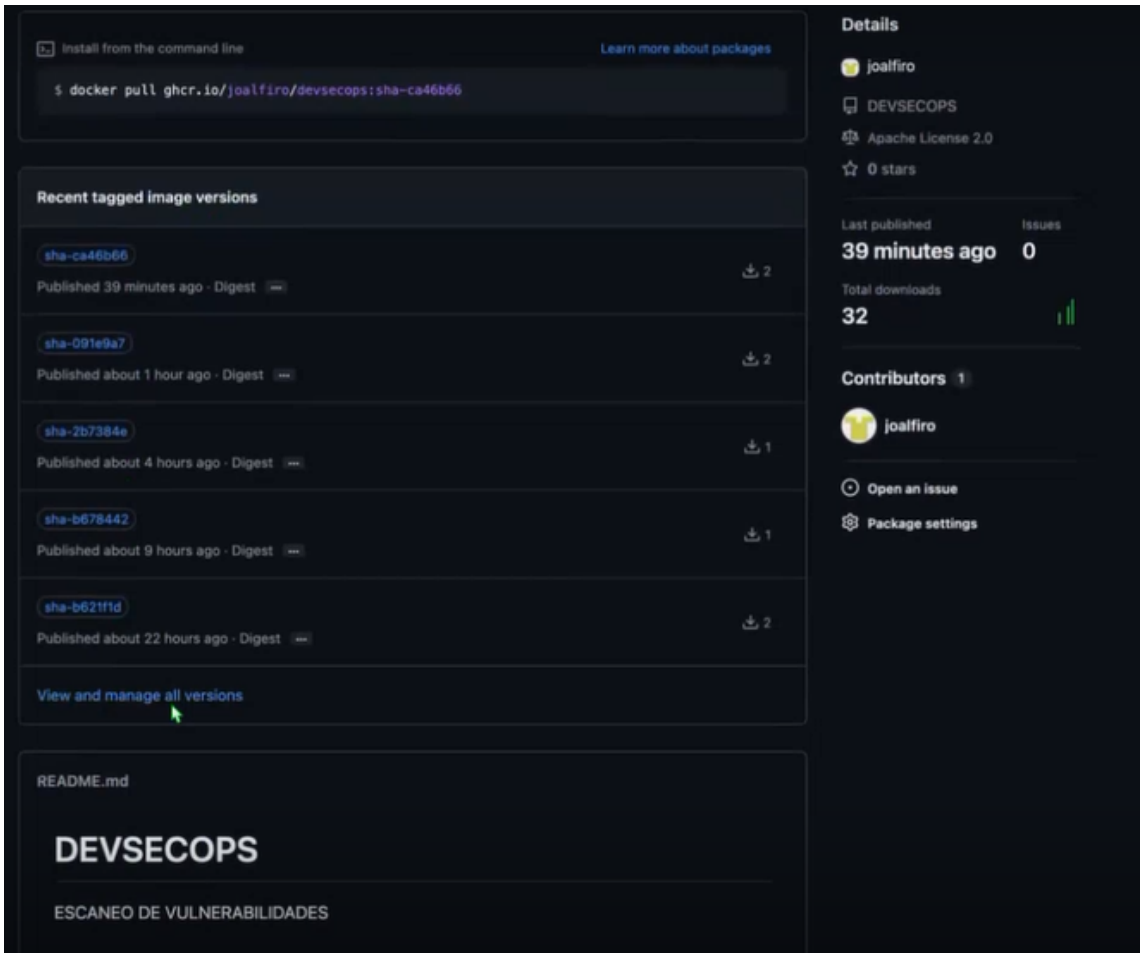


Figura 133. Múltiples Versiones Etiquetadas del Paquete “DEVSECOPS” en GitHub Container Registry

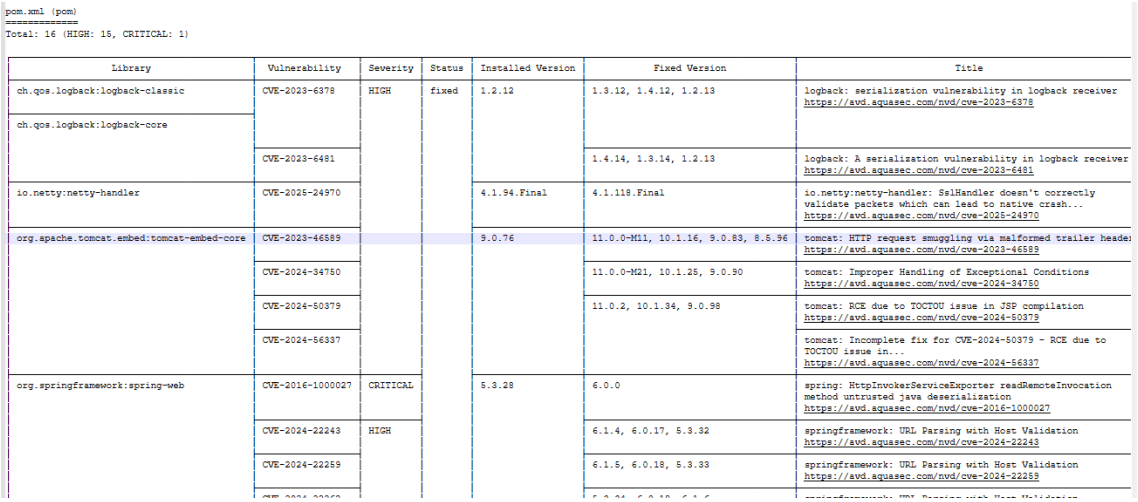
4.2 Resultados de la implementación de la Fase 2: Definición de procesos para la gestión de vulnerabilidades detectadas

En la Fase 2 se implementaron procesos para la gestión de vulnerabilidades en el core bancario, basándose en una clasificación según el potencial impacto en la seguridad del sistema (críticas, altas, medias y bajas). Con base en esta clasificación, se asignaron prioridades para abordar las vulnerabilidades, se implementó un plan integral de mitigación de riesgos y se generó un informe final que documenta tanto las incidencias detectadas como las acciones correctivas adoptadas. El objetivo de esta fase fue proveer información sobre el avance en la solución de las vulnerabilidades identificadas y establecer una guía práctica para la gestión de vulnerabilidades en el microservicio del laboratorio.

4.2.1 Antecedentes

Previo al pase, se realizó el escaneo de las imágenes, en el cual se evidenciaron vulnerabilidades clasificadas como críticas y altas, las cuales debieron ser solventadas para garantizar la seguridad y estabilidad del software. A continuación, se presentan los resultados obtenidos en esta fase, los cuales evidencian la efectividad del enfoque implementado y ofrecen una visión detallada de la situación de seguridad en el entorno estudiado.

La Figura 135 muestra una parte del informe de análisis de dependencias obtenido al escanear el archivo pom.xml de un proyecto bancario. En esta captura, se listan varias bibliotecas utilizadas por el microservicio, junto con las vulnerabilidades asociadas, su severidad, el estado actual (por ejemplo, “found” o “fixed”), las versiones instaladas y las recomendadas para corregir dichas incidencias. Este tipo de reporte es fundamental para priorizar las actualizaciones de componentes críticos y mantener la integridad del software en entornos que requieren altos estándares de seguridad.



pom.xml (pom)
==========
Total: 16 (HIGH: 15, CRITICAL: 1)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
ch.qos.logback:logback-classic	CVE-2023-6378	HIGH	fixed	1.2.12	1.3.12, 1.4.12, 1.2.13	logback: serialization vulnerability in logback receiver https://avd.aquasec.com/nvd/cve-2023-6378
ch.qos.logback:logback-core						
	CVE-2023-6481				1.4.14, 1.3.14, 1.2.13	logback: A serialization vulnerability in logback receiver https://avd.aquasec.com/nvd/cve-2023-6481
io.netty:netty-handler	CVE-2025-24970			4.1.94.Final	4.1.118.Final	io.netty:netty-handler: SslHandler doesn't correctly validate packets which can lead to native crash... https://avd.aquasec.com/nvd/cve-2025-24970
org.apache.tomcat.embed:tomcat-embed-core	CVE-2023-46589			9.0.76	11.0.0-M11, 10.1.16, 9.0.83, 8.5.96	tomcat: HTTP request smuggling via malformed trailer header https://avd.aquasec.com/nvd/cve-2023-46589
	CVE-2024-34750				11.0.0-M21, 10.1.25, 9.0.90	tomcat: Improper Handling of Exceptional Conditions https://avd.aquasec.com/nvd/cve-2024-34750
	CVE-2024-50379				11.0.2, 10.1.34, 9.0.98	tomcat: RCE due to TOCTOU issue in JSP compilation https://avd.aquasec.com/nvd/cve-2024-50379
	CVE-2024-56337					tomcat: Incomplete fix for CVE-2024-50379 - RCE due to TOCTOU issue in... https://avd.aquasec.com/nvd/cve-2024-56337
org.springframework:spring-web	CVE-2016-1000027	CRITICAL		5.3.28	6.0.0	spring: HttpInvokerServiceExporter readRemoteInvocation method untrusted java deserialization https://avd.aquasec.com/nvd/cve-2016-1000027
	CVE-2024-22243	HIGH			6.1.4, 6.0.17, 5.3.32	springframework: URL Parsing with Host Validation https://avd.aquasec.com/nvd/cve-2024-22243
	CVE-2024-22259				6.1.5, 6.0.18, 5.3.33	springframework: URL Parsing with Host Validation https://avd.aquasec.com/nvd/cve-2024-22259

Figura 134. Resultado de Escaneo de Vulnerabilidades en el Archivo pom.xml

El informe evidencia la presencia de librerías como ch.qos.logback, io.netty, y springframework, cada una con vulnerabilidades catalogadas según su nivel de gravedad y la respectiva versión que mitiga el problema. Algunas entradas muestran un estado de “found”, lo cual indica que se detectó la falla en la versión instalada; otras, en cambio, reflejan un estado de “fixed” cuando la dependencia ha sido actualizada o no se ve afectada. El uso de identificadores CVE (Common Vulnerabilities and Exposures) facilita

la trazabilidad de cada vulnerabilidad y su seguimiento en bases de datos de seguridad reconocidas. Este hallazgo resalta la importancia de implementar escaneos periódicos en el ciclo de para asegurar que las dependencias se mantengan en versiones libres de fallos críticos o altos, garantizando así la estabilidad y confiabilidad del microservicio.

En la siguiente tabla, se expone un conjunto de vulnerabilidades identificadas en las principales librerías y frameworks utilizados por el microservicio bancario, organizadas según su ID (CVE), el título o descripción breve, la versión afectada y la severidad reportada. Este listado resulta esencial para priorizar las acciones de remediación, especialmente en un entorno donde la seguridad y la confiabilidad del software son de suma importancia.

Tabla 15
Listado de Vulnerabilidades Detectadas en Dependencias Principales

ID	TITLE	VERSION	SEVERITY
CVE-2023-6378	ch.qos.logback:logback-classic	1.2.12	HIGH
CVE-2018-10054	ch.qos.logback:logback-classic	1.2.12	HIGH
CVE-2016-1000027	org.springframework:spring-web	5.3.28	CRITICAL
CVE-2024-22243	org.springframework:spring-web	5.3.28	HIGH
CVE-2023-6481	serialization vulnerability in logback receiver	1.2.12	HIGH
CVE-2025-24970	io.netty:netty-handler	4.1.94.Final	HIGH
CVE-2023-46589	org.apache.tomcat.embed:tomcat-embed-core	9.0.76	HIGH
CVE-2024-34750	Tomcat	9.0.76	HIGH
CVE-2024-50379	Tomcat	9.0.76	HIGH
CVE-2024-56337	Tomcat	9.0.76	HIGH
CVE-2024-22259	springframework: URL	5.3.28	HIGH
CVE-2024-22262	springframework: URL	5.3.28	HIGH
CVE-2024-38816	org.springframework:spring-webmvc	5.3.28	HIGH
CVE-2024-38819	org.springframework:spring-webmvc:	5.3.28	HIGH
CVE-2022-1471	org.yaml:snakeyaml	1.30	HIGH
CVE-2022-25857	org.yaml:snakeyaml	1.30	HIGH

Los datos evidencian la presencia de múltiples vulnerabilidades clasificadas como HIGH e incluso CRITICAL, que afectan componentes clave como ch.qos.logback:logback-classic, org.springframework:spring-web, io.netty, y Tomcat. La reiteración de CVEs asociados a logback y springframework subraya la necesidad de mantener actualizados los componentes de logging y los módulos de Spring para prevenir posibles ataques que podrían comprometer la integridad del sistema. Asimismo, la existencia de varias referencias a versiones afectadas de Tomcat resalta la relevancia de corregir o actualizar este servidor de aplicaciones, dado su uso frecuente en entornos bancarios. En conjunto, esta información reafirma la importancia de integrar escaneos periódicos y planes de mitigación en el ciclo de vida del desarrollo, garantizando que las dependencias críticas se mantengan libres de vulnerabilidades que pongan en riesgo la operación y la confidencialidad de los servicios financieros.

4.2.2 Resumen de actividades

La tabla 16 detalla un conjunto de vulnerabilidades identificadas en librerías y frameworks esenciales para el microservicio bancario, incluyendo logback, Spring, Netty, y Tomcat, todas con un estatus pendiente de corrección. En este listado se especifica el ID (CVE), el título, la versión afectada, el nivel de severidad (HIGH o CRITICAL) y la condición actual de cada incidencia. El objetivo de presentar estos datos es subrayar la necesidad de priorizar las actualizaciones y parches que mitiguen el riesgo en el entorno bancario.

Tabla 16
Resumen de Vulnerabilidades Pendientes en Dependencias Principales

ID	TITLE	VERSION	SEVERITY	ESTATUS
CVE-2023-6378	ch.qos.logback:logback-classic	1.2.12	HIGH	PENDIENTE
CVE-2018-10054	ch.qos.logback:logback-classic	1.2.12	HIGH	PENDIENTE
CVE-2016-1000027	org.springframework:spring-web	5.3.28	CRITICAL	PENDIENTE
CVE-2024-22243	org.springframework:spring-web	5.3.28	HIGH	PENDIENTE
CVE-2023-6481	serialization vulnerability in logback receiver	1.2.12	HIGH	PENDIENTE
CVE-2025-24970	io.netty:netty-handler	4.1.94.Final	HIGH	PENDIENTE
CVE-2023-46589	org.apache.tomcat.embed:tomcat-embed-core	9.0.76	HIGH	PENDIENTE

ID	TITLE	VERSION	SEVERITY	ESTATUS
CVE-2024-34750	Tomcat	9.0.76	HIGH	PENDIENTE
CVE-2024-50379	Tomcat	9.0.76	HIGH	PENDIENTE
CVE-2024-56337	Tomcat	9.0.76	HIGH	PENDIENTE
CVE-2024-22259	springframework: URL	5.3.28	HIGH	PENDIENTE
CVE-2024-22262	springframework: URL	5.3.28	HIGH	PENDIENTE
CVE-2024-38816	org.springframework:spring-webmvc	5.3.28	HIGH	PENDIENTE
CVE-2024-38819	org.springframework:spring-webmvc:	5.3.28	HIGH	PENDIENTE
CVE-2022-1471	org.yaml:snakeyaml	1.30	HIGH	PENDIENTE
CVE-2022-25857	org.yaml:snakeyaml	1.30	HIGH	PENDIENTE

Los resultados muestran que la totalidad de las vulnerabilidades se encuentra en estado PENDIENTE, evidenciando la urgencia de aplicar soluciones antes de avanzar con el despliegue en producción. Llama la atención la presencia de varias entradas relacionadas con Spring (5.3.28) y Tomcat (9.0.76), componentes ampliamente utilizados en aplicaciones bancarias que, de no ser corregidos, podrían exponer la plataforma a amenazas críticas. Asimismo, el hecho de que algunas vulnerabilidades datan de versiones anteriores (por ejemplo, CVE-2016-1000027 o CVE-2018-10054) refuerza la importancia de un monitoreo continuo y una estrategia de actualización que evite la acumulación de fallas no atendidas. Estas evidencias resaltan la pertinencia de un enfoque DevSecOps, donde la detección temprana de vulnerabilidades se integre con planes de mitigación ágiles y efectivos, salvaguardando así la integridad y disponibilidad de los servicios financieros.

Para poder visualizar las dependencias de las componentes que se utilizan en la aplicación se utiliza el siguiente comando mvn dependece:tree . Además, Es importante verificar cual es el componente padre de la librería utilizada que tiene vulnerabilidades para poder verificar si existe alguna solución por parte del proveedor sobre la vulnerabilidad, en este escenario la componente padre es la librería de spring-boot versión 2.7.13 como se muestra en el árbol de dependencias obtenido.

Figura 135. Árbol de dependencias

El despliegue de dependencias confirma la presencia de módulos críticos como org.springframework.boot, org.apache.tomcat.embed, io.netty y com.fasterxml.jackson, que se han identificado en los análisis previos como susceptibles a fallas de seguridad clasificadas con severidad alta o crítica. Esto pone de relieve la necesidad de mantener un proceso de actualización continuo y de supervisar el estado de cada librería, integrando herramientas de escaneo (por ejemplo, Trivy, OWASP Dependency-Check) en el pipeline de CI/CD. Asimismo, la inclusión de dependencias de prueba (test) indica un enfoque de validación que, cuando se combina con la adopción de prácticas DevSecOps, facilita la detección temprana de vulnerabilidades y la rápida implementación de parches o correcciones en entornos bancarios.

De acuerdo a la documentación de Maven en la librería de spring boot versión 2.7.13, se han detectado varias vulnerabilidades, como lo indica la figura a continuación:

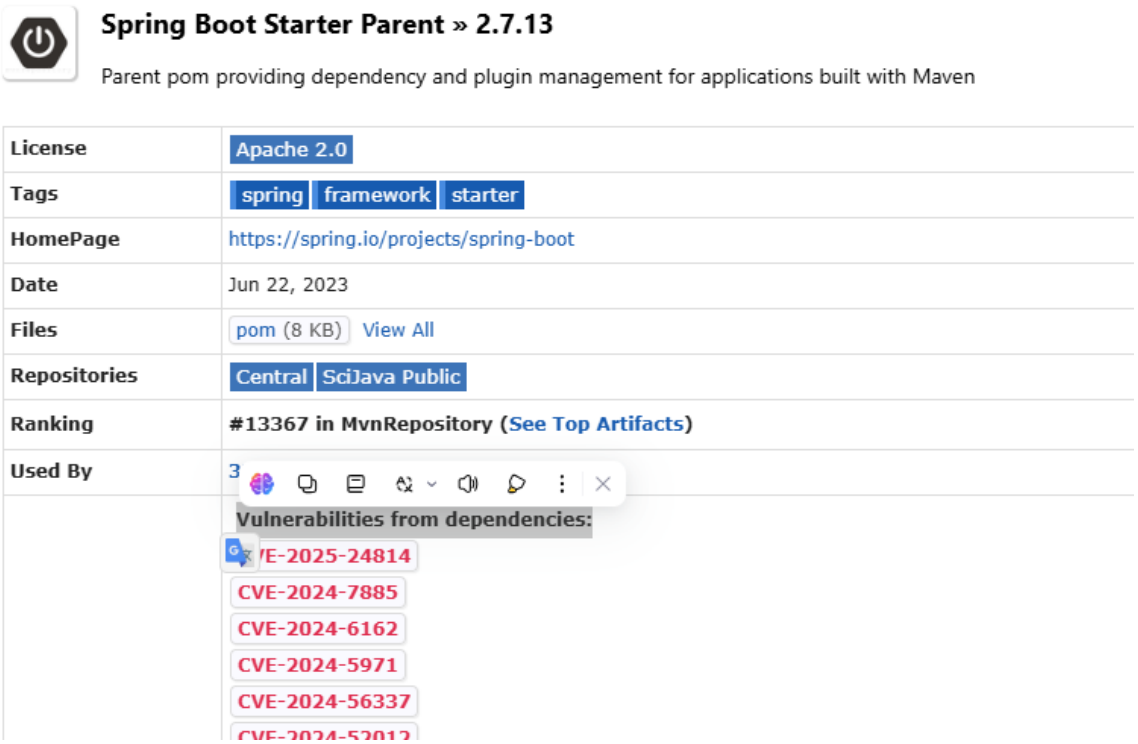


Figura 136. Vulnerabilidades Asociadas a Spring Boot Starter Parent 2.7.13

Ref: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-parent/2.7.13

La imagen confirma la existencia de vulnerabilidades críticas y altas, asociadas a componentes subyacentes en el ecosistema de Spring Boot. Entre los CVE destacados, se observan referencias a versiones específicas que requieren actualización o parche para

mitigar riesgos. El hecho de que se muestren múltiples CVE para una sola versión de Spring Boot Starter Parent enfatiza la relevancia de adoptar prácticas de verificación continua de dependencias y parches. En el contexto de la banca, esta evidencia respalda la importancia de integrar un pipeline DevSecOps, donde cada actualización de Spring Boot se somete a un escrutinio sistemático para asegurar la corrección de vulnerabilidades antes de implementar los servicios en entornos de producción, y se recomienda utilizar una versión más nueva de la componente para solventar estas vulnerabilidades. Para este escenario se procedió a actualizar la versión de spring-boot a la última componente estable del proveedor en este caso la versión 3.4.3

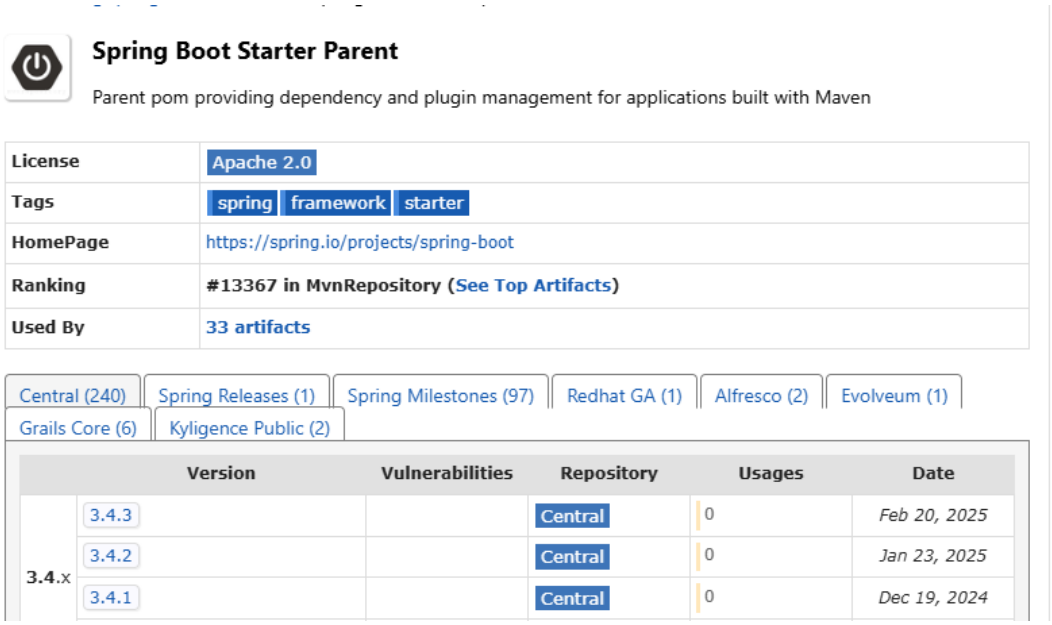


Figura 137. Detalles Generales del Spring Boot Starter Parent en Maven

ref: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-parent

La imagen resalta la licencia de código abierto (Apache 2.0) y la clasificación del proyecto (Spring, framework, starter), elementos cruciales al evaluar el cumplimiento normativo y la compatibilidad con otras librerías. Adicionalmente, se observa que el artefacto se encuentra en un nivel de popularidad considerable (Ranking #13367 en MvnRepository), indicando una adopción amplia en diversos proyectos. Sin embargo, la simple presencia en un repositorio de alta visibilidad no exime la necesidad de monitorear continuamente las vulnerabilidades reportadas, como se aprecia en los análisis previos, donde distintas versiones de Spring Boot Starter Parent presentan fallas clasificadas con severidad alta o crítica. En un entorno bancario, este panorama refuerza la importancia de

realizar escaneos periódicos y de mantener una estrategia de actualización y parches que minimice el riesgo de compromisos de seguridad.

Se actualizo la versión de java y spring boot en el Pom de la aplicación antes se tenía la versión Java 11 spring-boot-starter-parent 2.7.13 y se procede con la actualización a Java 17 spring-boot-starter-parent 3.4.3.

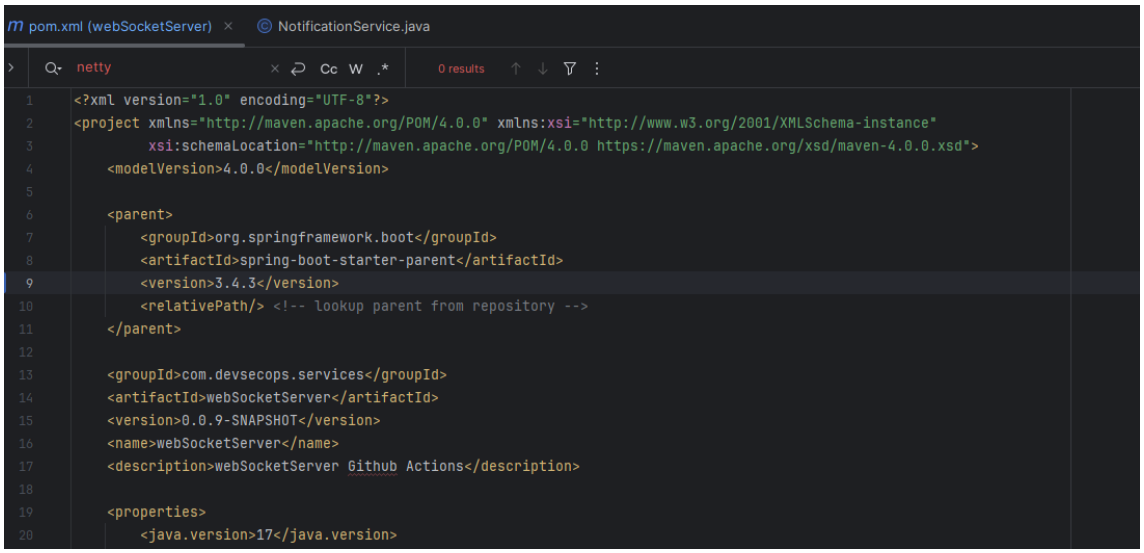


Figura 138. Fragmento del Archivo pom.xml con Referencia al Spring Boot Starter Parent

Este snippet refuerza la relevancia de mantener actualizadas las dependencias y alinea el desarrollo con la adopción de un pipeline seguro y estable. Luego de la subir los cambios a una rama de desarrollo se genera una nueva imagen la cual se verifica el nuevo escaneo de vulnerabilidades (ghcr.io/joalfiro/devsecops:sha-bbfabdd). Después del escaneo realizado se solventaron todas las vulnerabilidades de las componentes, tal como lo muestra la Figura 137

Figura 139. Mensaje de Trivy con VEX Notice y Resultado de Escaneo

El mensaje indica que, según el escaneo realizado, no se han encontrado vulnerabilidades graves o críticas en la imagen del contenedor. No obstante, Trivy advierte a los mantenedores de proyectos open source que, en caso de identificar fallas no explotables o que requieran un contexto específico para su activación, puede considerarse la publicación de un VEX statement. Esta funcionalidad aporta transparencia y seguridad adicional al proceso DevSecOps, al permitir que los equipos declaren públicamente las condiciones en las cuales una vulnerabilidad no supone un riesgo real.

Además, el trivy también realiza un escaneo del software base utilizado alpine 3.21.3 para generar la imagen la cual tampoco presenta vulnerabilidades.

4.2.3 Análisis del Último Reporte Generado

Una vez realizada la actualización de los componentes detectados con vulnerabilidades se pudo observar que las mismas fueron solventadas por el proveedor

Tabla 17
Vulnerabilidades Resueltas en Dependencias Críticas

ID	TITLE	VERSION	SEVERITY	ESTATUS
CVE-2023-6378	ch.qos.logback:logback-classic	1.2.12	HIGH	Resuelta
CVE-2018-10054	ch.qos.logback:logback-classic	1.2.12	HIGH	Resuelta
CVE-2016-1000027	org.springframework:spring-web	5.3.28	CRITICAL	Resuelta
CVE-2024-22243	org.springframework:spring-web	5.3.28	HIGH	Resuelta
CVE-2023-6481	serialization vulnerability in logback receiver	1.2.12	HIGH	Resuelta
CVE-2025-24970	io.netty:netty-handler	4.1.94.Final	HIGH	Resuelta
CVE-2023-46589	org.apache.tomcat.embed:tomcat-embed-core	9.0.76	HIGH	Resuelta
CVE-2024-34750	Tomcat	9.0.76	HIGH	Resuelta
CVE-2024-50379	Tomcat	9.0.76	HIGH	Resuelta
CVE-2024-56337	Tomcat	9.0.76	HIGH	Resuelta
CVE-2024-22259	springframework: URL	5.3.28	HIGH	Resuelta
CVE-2024-22262	springframework: URL	5.3.28	HIGH	Resuelta
CVE-2024-38816	org.springframework:spring-webmvc	5.3.28	HIGH	Resuelta
CVE-2024-38819	org.springframework:spring-webmvc:	5.3.28	HIGH	Resuelta

ID	TITLE	VERSION	SEVERITY	ESTATUS
CVE-2022-1471	org.yaml:snakeyaml	1.30	HIGH	Resuelta
CVE-2022-25857	org.yaml:snakeyaml	1.30	HIGH	Resuelta

El análisis de la información muestra que se han solucionado vulnerabilidades previamente clasificadas como HIGH o CRITICAL en componentes clave, como ch.qos.logback, Spring Web, Tomcat, Netty y snakeyaml. El hecho de que todas figuren con un estatus de "Resuelta" indica la implementación exitosa de parches, actualizaciones o configuraciones que subsanan las fallas detectadas. Este logro pone de relieve la efectividad de las prácticas DevSecOps, en las que la identificación temprana de vulnerabilidades y la adopción de planes de mitigación resultan fundamentales para garantizar la seguridad en entornos bancarios. Asimismo, la capacidad de rastrear y documentar cada CVE denota una adecuada trazabilidad, factor indispensable en la conformidad con normas internas y externas, además de reforzar la confianza en la fiabilidad del software desplegado.

4.3 Descripción del Caso de Estudio: Integración de un microservicio del backend de un core bancario a la plataforma de pago de servicios de terceros.

4.3.1 Justificación en la elección del caso de estudio

4.3.1.1. Relevancia de la Integración con Terceros en el Contexto Bancario.

En la banca, los procesos de pago a terceros (servicios públicos, compañías de seguros, recaudaciones municipales, etc.) forman parte esencial de las funcionalidades ofrecidas a los clientes. El módulo de pago de servicios externos constituye un caso realista y habitual en las plataformas bancarias, pues integra el flujo financiero con entidades que manejan datos sensibles de los usuarios (nombres, direcciones, información de consumo o incluso detalles bancarios para pagos en línea).

4.3.1.2. Simulación con Datos Sensibles y Entorno Regulatorio.

En el pago de servicios se maneja datos personales y financieros que se alinean con el tipo de información crítica propia de un core bancario:

- Información personal (nombres, direcciones, teléfonos).
- Historial de consumos (análogo a un historial de transacciones bancarias).
- Información de pagos en línea (equivalente a la gestión de tarjetas o débitos bancarios).

En consecuencia, la simulación de un microservicio que interactúe con un servicio externo replica de manera fidedigna los requerimientos de seguridad, confidencialidad y cumplimiento normativo que se exigen en entornos bancarios, incluida la Ley Orgánica de Protección de Datos Personales (LOPDP) en Ecuador.

4.3.1.3. Transferencia de Conocimientos y Metodologías.

La metodología DevSecOps y las herramientas open source presentadas son plenamente aplicables a cualquier microservicio que procese datos sensibles y deba cumplir con regulaciones de protección de la información.

El caso de estudio sobre un microservicio del core bancario para la integración del pago de servicios con terceros permite demostrar cómo se adapta la estrategia DevSecOps a un escenario concreto de integración con terceros, validando la robustez del enfoque (automatización de pruebas, detección de vulnerabilidades, gestión de parches, etc.) y reforzando el cumplimiento de la LOPDP y otros lineamientos de seguridad.

4.3.1.4. Casuística y Complejidad Operativa.

La selección de este caso de estudio atiende a la complejidad que conlleva un servicio de recaudación externo en un ecosistema bancario. La simulación de un pago de servicios de terceros ilustra una arquitectura distribuida donde los microservicios deben comunicarse con un proveedor que mantiene su propia infraestructura y maneja datos críticos. Esto reproduce de forma fiel la problemática de los bancos al integrar soluciones

de terceros, lo cual conlleva la necesidad de garantizar la integridad de la información, la autenticación de usuarios y la disponibilidad de los sistemas.

4.3.1.5. Convergencia con los Objetivos.

La elección de un microservicio que simula el pago de servicios de terceros es representativa de la clase de transacciones que se gestionan a diario en los entornos bancarios. Este caso permite validar los objetivos planteados en la tesis con un ejemplo práctico que comparte las mismas exigencias y restricciones de un core bancario.

4.3.1.6. Factibilidad y Disponibilidad de la Información.

Por último, la elección del caso de estudio responde también a la factibilidad de disponer de un escenario de prueba donde se manejan datos sensibles y se cumplen procesos de recaudación. En muchos casos, los entornos bancarios son de acceso restringido por su criticidad y sujeta a acuerdos de confidencialidad. Al contar con la colaboración o simulación de un servicio externo, se obtiene un caso de estudio viable y suficientemente robusto para demostrar la eficacia de DevSecOps sin exponer la infraestructura bancaria.

4.3.2 Descripción del Caso de estudio

El caso de estudio está dirigido a la aplicación de DevSecOps en la integración de un microservicio para el pago de servicios de terceros, para esto se ha elegido un ejemplo de una simulación de la respuesta del webservice de un servicio externo, ya que expone un microservicio para la gestión y recaudación del servicio, en una ciudad de tamaño medio. La empresa maneja datos sensibles, incluyendo:

- Información personal de los usuarios (nombres, direcciones, teléfonos).
- Historial de consumo.
- Información bancaria de pagos en línea, entre otros.

Hemos identificado una falta de integración de la seguridad en el ciclo de desarrollo de software. Los recientes ataques cibernéticos y normativas regulatorias nos han motivado

la adopción de DevSecOps para mejorar la seguridad y la agilidad en la entrega de software.

La siguiente figura ilustra el flujo de interacción entre el usuario frontend, el microservicio WebSocketServer y un Dummy que simula la respuesta del sistema externo y el uso de Redis como servicio de apoyo. En este esquema se detalla cómo, al recibir una petición desde el canal de comunicación, el WebSocketServer establece una conexión bidireccional vía WebSocket, procesando la información y enviándola posteriormente al core bancario (representado en este caso por un dummy de prueba) para su validación y respuesta.

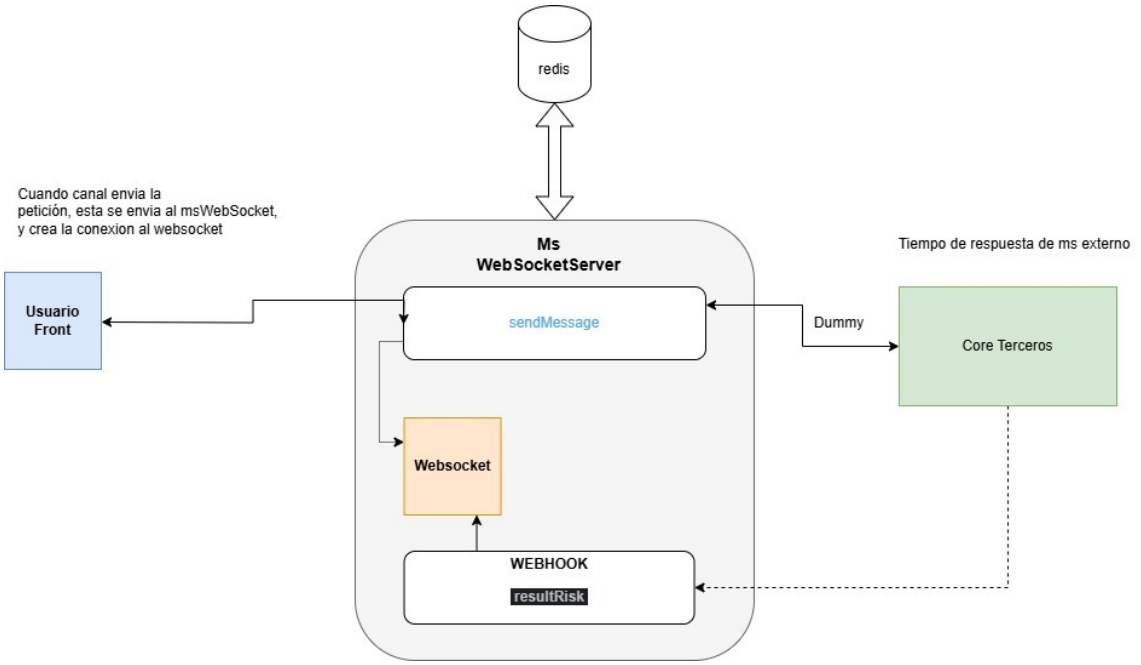


Figura 140. Diagrama de Integración del Microservicio WebSocketServer con Redis y Core Terceros

El diagrama evidencia que el microservicio WebSocketServer actúa como un punto central para gestionar las comunicaciones en tiempo real, recibiendo solicitudes desde la interfaz de usuario y despachándolas hacia el sistema central. La integración de Redis sugiere la posibilidad de mantener un almacenamiento en memoria o una capa de mensajería intermedia para mejorar la escalabilidad y la rapidez en la gestión de eventos. Asimismo, la presencia de un Webhook (resultRisk) indica la capacidad de notificar a otros servicios sobre los resultados del proceso de validación, reforzando la idea de un ecosistema de microservicios desacoplados. De este modo, la arquitectura propuesta

ofrece un canal asíncrono y escalable para las operaciones críticas, algo fundamental en un contexto bancario o de servicios que requieren baja latencia y alta disponibilidad.

4.3.2.1. Problemas Identificados.

1. No adopción de DevSecOps: depende de procesos manuales para la integración continua de un microservicio y el análisis de vulnerabilidades en el código fuente.
2. Baja seguridad: Los procesos de seguridad son reactivos y no están integrados en el ciclo de desarrollo.
3. Lentitud en la entrega de nuevas funcionalidades: Las actualizaciones requieren semanas o meses debido a procesos manuales de pruebas y despliegue.
4. Cumplimiento normativo: no cumplimiento de los principios de la ley Orgánica de Protección de Datos Personales (LOPDP) del Ecuador.

4.3.2.2. Objetivos.

1. Automatizar el ciclo de desarrollo y despliegue de software.
2. Incorporar seguridad desde el inicio del ciclo de vida del desarrollo de software (SDLC).
3. Mitigar vulnerabilidades y garantizar la protección de datos de los usuarios.
4. Mejorar la colaboración entre los equipos de desarrollo, operaciones y seguridad.

4.3.2.3. Metodología DevSecOps Aplicada.

1. Evaluación inicial

- Realizar un diagnóstico de la infraestructura actual.
- Identificar herramientas de automatización y seguridad necesarias.
- Capacitar a los equipos en prácticas DevSecOps.

2. Integración de herramientas

- Planificación: GitHub Dashboard
- Versionamiento del código: GitHub

- Compilación de código: Apache maven
- Testing de código: SonarQube
- CI/CD (Release/Deploy): GitHub Actions
- Operaciones: Docker
- Monitoreo: Grafana
- Detección de vulnerabilidades: Trivy

3. *Control de accesos y monitoreo*

    - Configurar herramientas de monitoreo como grafana para detectar actividades sospechosas mediante la revisión de las métricas del uso de memoria, cpu.
    - Implementar uid_entrypoint para validar si la aplicación es vulnerable para usuarios y password dentro del sistema operativo de la imagen del contenedor.

4. *Cultura colaborativa*

    - Establecer un flujo de trabajo donde los equipos de desarrollo, operaciones y seguridad trabajen en conjunto.
    - Realizar revisiones de código conjuntas y sesiones de retroalimentación constantes.
    - Implementar GitHub Dashboard para gestionar actividades en el equipo y fomentar la seguridad como responsabilidad compartida.

5. *Pruebas continuas*

    - Implementar la herramienta Trivy de seguridad de código abierto que analiza contenedores Docker y dependencias en busca de vulnerabilidades.
    - Implementar la herramienta SonarQube de análisis estático de código que evalúa la calidad del código fuente en busca de errores, vulnerabilidades y malas prácticas.
    - Introducir alertas tempranas para detectar posibles vulnerabilidades.
    - Cumplir con la cobertura de pruebas de testing mediante la implementación de la herramienta sonarQube.

4.3.2.4. Resultados Esperados.

La siguiente figura representa el diagrama de flujo que describe cómo el código fuente se gestiona desde su repositorio, se somete a validaciones y análisis de seguridad, y finalmente se empaqueta en contenedores para su despliegue. En este esquema, GitHub Actions funge como orquestador principal del pipeline, ejecutando procesos de construcción (build) con Maven, análisis estático (SonarQube), escaneo de imágenes (Trivy) y generación de contenedores Docker antes de ser trasladados a los entornos de ejecución correspondientes.

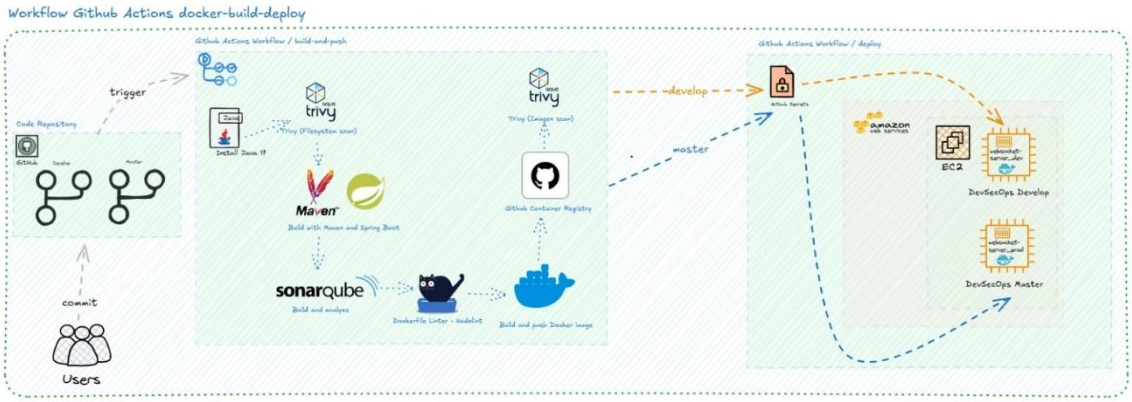


Figura 141. Flujo de Trabajo con GitHub Actions para Construcción y Despliegue de Contenedores

El diagrama evidencia un pipeline en el que los desarrolladores efectúan commits en el repositorio, desencadenando una serie de acciones automáticas: en primer lugar, Maven se encarga de compilar el proyecto y gestionar dependencias, seguido de SonarQube para identificar vulnerabilidades o malas prácticas en el código fuente (SAST). Luego, Trivy revisa las imágenes Docker generadas para detectar vulnerabilidades en las capas de contenedor. Si los resultados cumplen con los umbrales de calidad y seguridad establecidos, GitHub Actions procede a empaquetar el contenedor y enviarlo al entorno de despliegue (master o producción), cerrando así el ciclo de entrega continua. Este flujo integral, que une el control de código, la verificación de seguridad y la automatización de despliegues, ejemplifica la adopción de un enfoque DevSecOps, donde la seguridad se incorpora de manera temprana y constante a lo largo del ciclo de vida del desarrollo.

1. Adopción de DevSecOps: aplicación de DevSecOps en la integración de un microservicio para el pago de servicios de terceros.

2. Arquitectura basada en microservicios, WebSockets y webhooks para una comunicación eficiente.
3. Reducción de vulnerabilidades: Implementación de un pipeline CI/CD con análisis de seguridad en cada etapa.
4. Eficiencia en despliegues: Tiempo de entrega de nuevas funcionalidades reducido a días o incluso horas.
5. Cumplimiento normativo: Auditorías de seguridad y conformidad automatizadas.
6. Mayor confianza del cliente: Protección de datos personales y financieros de los usuarios finales.

4.3.2.5. Indicadores Clave de Desempeño.

1. Reducción de incidentes de seguridad: Monitorear los eventos de seguridad antes y después de implementar DevSecOps.
2. Tiempo de despliegue: Medir el tiempo promedio para implementar cambios en producción.
3. Porcentaje de cumplimiento normativo: Validar auditorías y evaluaciones externas.
4. Satisfacción del cliente: Evaluar encuestas y entrevistas relacionadas a la implementación de prácticas DevSecOps y el uso de herramientas open source orientadas a microservicios.

4.4 Resultados de la implementación de la Fase 3: Análisis del nivel de cumplimiento de la aplicación de DevSecOps

En la Fase 3 se llevó a cabo el análisis del nivel de cumplimiento en la aplicación de DevSecOps, tomando como referencia la norma ISO/IEC 27034 para evaluar las vulnerabilidades encontradas a lo largo del ciclo de vida del desarrollo de software bancario. A partir de este estudio, se elaboró una matriz de resultados que identifica las brechas existentes y las áreas de mejora, al mismo tiempo que ofrece directrices claras para reforzar la seguridad de las aplicaciones. A continuación, se exponen los hallazgos y conclusiones derivados de este proceso, con el propósito de ilustrar el estado actual de conformidad y proponer las acciones necesarias para elevar el nivel de protección en el entorno bancario.

4.4.1 Matriz de Análisis de Cumplimiento de DevSecOps según ISO/IEC 27034

Analizar el nivel de cumplimiento de la aplicación de DevSecOps de acuerdo con la norma ISO/IEC 27034 en el proceso de desarrollo de software.

4.4.1.1. Información General.

Esta matriz permite evaluar el estado de la implementación de DevSecOps en relación con la norma ISO/IEC 27034, identificar brechas y establecer planes de mejora.

4.4.1.2. Instrucciones.

- Para cada cláusula y requisito específico de la norma ISO/IEC 27034, identificamos la práctica DevSecOps asociada que se aplicó en el caso de estudio (Microservicio Core Bancario para la integración del pago de servicios con terceros).
- Recopilamos evidencia que demuestra el cumplimiento de cada práctica DevSecOps.
- Evalúanos el nivel de cumplimiento (Alto, Medio o Bajo) para cada práctica DevSecOps, basándonos en la evidencia recopilada.
- Registramos observaciones y recomendaciones para mejorar el cumplimiento de la norma ISO/IEC 27034 en el proceso de desarrollo de software.
- Recursos Adicionales:
- Norma ISO/IEC 27034
- Guías de DevSecOps

La siguiente tabla presenta una matriz de cumplimiento que asocia los requisitos específicos de la norma ISO/IEC 27034 con las prácticas DevSecOps implementadas en el caso de estudio. En ella se documentan las evidencias de cumplimiento, el nivel alcanzado (Alto, Medio o Bajo) definido por la norma y las observaciones correspondientes fueron valoradas en una mesa de trabajo conformada por el equipo de investigación que desarrollo la tesis. Esta matriz permite evaluar, de forma estructurada, la integración de la seguridad a lo largo del ciclo de vida del desarrollo del software

bancario, identificando las áreas con un mayor grado de conformidad y aquellas que requieren planes de mejora adicionales.

Tabla 18.
Matriz de Cumplimiento DevSecOps Según ISO/IEC 27034

Cláusula de ISO/IEC 27034	Requisito Específico	Práctica DevSecOps Asociada	Evidencia de Cumplimiento	Nivel de Cumplimiento (Alto/Medio/Bajo)	Observaciones/Evidencias
5.1 Establecimiento del contexto de seguridad	Definir el alcance de la seguridad de las aplicaciones	Análisis de riesgos de seguridad de la aplicación	Documento de análisis de riesgos, políticas de seguridad	Alto	Implementamos la herramienta Trivy para el análisis de seguridad en contenedores, imágenes, repositorios de código y dependencias en busca de vulnerabilidades, y configuraciones incorrectas. Plan de integración de desarrollo seguridad y operaciones
5.2 Evaluación de riesgos de seguridad	Identificar y evaluar las amenazas y vulnerabilidades	Pruebas de seguridad estáticas y dinámicas (SAST/DAST), análisis de código	Informes de pruebas, resultados de análisis de vulnerabilidades	Medio	Informe de vulnerabilidades Implementamos la herramienta SonarQube para análisis de código estático, mejorando la calidad del código y la cobertura de las pruebas. Informe de validación del enfoque DevSecOps
5.3 Tratamiento de riesgos de seguridad	Seleccionar e implementar controles de seguridad	Integración de herramientas de seguridad en el CI/CD, gestión de vulnerabilidades	Registros de implementación de controles, políticas de seguridad actualizadas	Medio	Implementamos la herramienta Trivy y sonarQube para la gestión de vulnerabilidades y mediante la implementación de GitHub Dashboard que es un panel de control para gestionar el repositorio, colaboraciones y tareas pendientes y gestionar actividades en equipo. Informe de integración de herramientas Informe de vulnerabilidades Informe de validación del enfoque DevSecOps
6.1 Diseño de seguridad	Incorporar la seguridad en el diseño de la aplicación	Modelado de amenazas, diseño de arquitectura segura	Diagramas de arquitectura, documentación de diseño	Alto	Documentación detallada del pipeline Diagramas de arquitectura CasoDeUsoWebHook.drawio, DevSecOps.excalidraw. Caso de estudio
6.2 Desarrollo seguro	Implementar prácticas de codificación segura	Revisiones de código, análisis estático de código	Resultados de revisiones de código, informes de análisis estático	Alto	Utilizamos la plataforma GitHub para almacenar, gestionar y compartir código utilizando el sistema de control de versiones Git y mediante GitHub Actions se realizó la implementación de la herramienta SonarQube para el análisis estático de código que evalúa el código fuente en busca de errores, vulnerabilidades, malas prácticas y problemas de

Cláusula de ISO/IEC 27034	Requisito Específico	Práctica DevSecOps Asociada	Evidencia de Cumplimiento	Nivel de Cumplimiento (Alto/Medio/Bajo)	Observaciones/Evidencias
					calidad en el código fuente del proyecto ayudando a mejorar la seguridad, mantenibilidad y rendimiento del software. Informe de validación del enfoque DevSecOps
6.3 Pruebas de seguridad	Verificar la efectividad de los controles de seguridad	Pruebas de penetración, pruebas de seguridad funcionales	Informes de pruebas de penetración, resultados de pruebas de seguridad	Medio	Implementamos la herramienta SonarQube para mejorar la calidad de código y medir la cobertura de las pruebas unitarias de los métodos utilizados.
6.4 Implementación y despliegue seguros	Asegurar la configuración segura de la aplicación	Automatización de despliegue seguro, gestión de configuración	Scripts de despliegue, políticas de configuración	Medio	Utilizamos la herramienta GitHub Actions para crear flujos de trabajo (workflows) y realizar tareas automáticamente en respuesta a eventos dentro de un repositorio. Documentación detallada del pipeline
6.5 Operaciones y mantenimiento seguros	Monitorear y gestionar la seguridad de la aplicación en producción	Monitoreo de seguridad en tiempo real, gestión de incidentes	Registros de monitoreo, informes de incidentes	Alto	Implementamos la herramienta Grafana para visualización de datos y monitoreo en tiempo real y para crear dashboards interactivos con gráficos, alertas y paneles para analizar métricas de diferentes fuentes de datos. Utilizamos la plataforma de virtualización Docker para empaquetar la aplicación con todas sus dependencias en un entorno aislado y portable. Plan de integración de desarrollo seguridad y operaciones

La matriz evidencia un grado de cumplimiento Alto en cláusulas como el Establecimiento del contexto de seguridad y la Operación y mantenimiento seguros, lo que refleja la implementación satisfactoria de procesos de análisis de riesgos y monitoreo continuo a través de herramientas como Trivy, Grafana y flujos de trabajo automatizados en GitHub Actions. Por otro lado, se observan niveles de cumplimiento Medio en aspectos relacionados con la Evaluación de riesgos de seguridad, el Tratamiento de riesgos y las Pruebas de seguridad, lo que indica la necesidad de reforzar las pruebas de penetración y la integración de prácticas de seguridad más exhaustivas (por ejemplo, la adopción de análisis dinámico o la profundización en metodologías de threat modeling). Además, el uso de herramientas como SonarQube y Docker destaca la adopción de buenas prácticas

de codificación segura y despliegue automatizado, lo que contribuye a mantener una arquitectura de microservicios robusta y alineada con los principios de DevSecOps. En conjunto, estos resultados señalan un progreso significativo hacia la seguridad en el desarrollo bancario, a la vez que apuntan a oportunidades de mejora para alcanzar un mayor nivel de conformidad con la norma ISO/IEC 27034.

4.5 Resultados de la implementación de la Fase4: Validación del enfoque DevSecOps

En la Fase 4 se realizó el análisis de las métricas de seguridad en el caso de estudio, abordando dimensiones como defectos, eficiencia, mantenibilidad, portabilidad, viabilidad y seguridad, estas dimensiones de las métricas que SonarQube entrega son definidas en base a estándares de codificación específicos para cada lenguaje de programación, principios de diseño de software, normas de seguridad OWASP. Con este propósito, se elaboró un documento específico para revisar el código de las aplicaciones recientemente modificadas o desarrolladas, aplicando las correcciones necesarias a las vulnerabilidades más comunes detectadas durante el ciclo de vida de desarrollo del software bancario. Asimismo, se establecieron métricas de seguridad que permiten medir el impacto de la adopción de DevSecOps en la calidad y protección de las aplicaciones. De esta manera, la definición de un caso de estudio práctico respalda la validación del enfoque DevSecOps, evidenciando la efectividad de las prácticas implementadas y las mejoras alcanzadas en el entorno bancario.

Tabla 19
Registro de Modificaciones en el Proceso de Diseño y Entrega de Servicio

Revisión de Código.			Diseño y entrega de servicio Requerimiento #: 9 Versionamiento Master		
Responsable:	develop		Aplicativo: WebSocket		
Fecha creación:	21/02/2025		Validacion WebSocket		
Resumen VERSIONAMIENTO PRODUCCION 21-febrero-2025 Registro de modificaciones					
Versión	**Descripción**	**Autor**	**Fecha**	**Aprobado por**	**Fecha de aprobación**
1.0	Creación del documento	develop	21/02/2025		dd/mm/aaaa

La tabla anterior exhibe el registro de modificaciones del documento de diseño y entrega de servicio, así como la revisión de código para el requerimiento número 9 (versionamiento en la rama master). En ella se incluyen datos relevantes como la versión del documento, la descripción de los cambios realizados, el autor, la fecha de elaboración y la persona responsable de aprobar dichas modificaciones. De esta forma, la tabla asegura la trazabilidad de los ajustes efectuados y documenta la evolución del proyecto para futuros seguimientos o auditorías.

4.5.1 Objetivo:

Esta sección tiene como objetivo realizar la revisión de código a las aplicaciones modificadas y/o nuevas desarrolladas, aplicando soluciones que corrijan y/o mitiguen las vulnerabilidades más comunes encontradas durante el ciclo de vida de desarrollo del software.

4.5.2 Componentes y Programas Analizados

Se ha analizado la convención de valores para los campos que se describen a continuación:

Tipo Fuentes:

- Nuevo
- Modificado

Estado:

- Verificado: cumple con lo solicitado
- Corregido: no cumple con lo solicitado, pero fue corregido el defecto
- No aplica: no se debe ejecutar el análisis para el programa/componente
- Pendiente: por el impacto no se va a corregir y se asume el riesgo

4.5.3 Lista de programas/componentes WEB

La Tabla 20 presenta un componente específico del proyecto, indicando su nombre o ruta, el tipo de fuente, el lenguaje de programación utilizado y el estado en el que se encuentra.

Este registro facilita la trazabilidad de los elementos desarrollados y la verificación de las tecnologías aplicadas en el ciclo de vida del software bancario.

Tabla 20
Descripción de Componente/Programa

No	Nombre del Componente/Programa	Tipo de fuente	Lenguaje	Estado
1	ghcr.io/joalfiro/devsecops:sha-bbfabdd	Git	java	

En la tabla, se identifica repositorio o imagen alojada en Git, implementado en Java y que, de acuerdo con la clasificación. El uso de Git como "Tipo Fuente" sugiere que el desarrollo o empaquetado del componente se gestiona a través de un repositorio de control de versiones, lo que coincide con las prácticas de integración continua propias de un enfoque DevSecOps. Esta información resulta fundamental para el seguimiento y la gestión de cada microservicio o elemento que conforma el ecosistema de la aplicación bancaria.

4.5.4 Revisión de Código Sonar

La Tabla 21 muestra la información esencial de un componente verificado dentro del proyecto, describiendo su nombre o ruta, el tipo de fuente, el lenguaje de programación empleado y su estado actual. Este registro forma parte del proceso de control y seguimiento de cada pieza que integra la solución, facilitando la identificación y trazabilidad de los recursos implementados.

Tabla 21
Descripción de Componente/Programa

No	Nombre del Componente/Programa	Tipo de fuente	Lenguaje	Estado
1	ghcr.io/joalfiro/devsecops:sha-bbfabdd	DOKER	java	Verificado

El componente ghcr.io/joalfiro/devsecops:sha-bbfabdd se cataloga como DOCKER (Tipo Fuente), indicando que se trata de una imagen o contenedor almacenado en un repositorio Docker, programado en JAVA, y que ha sido verificado satisfactoriamente. Esta verificación sugiere que el contenedor cumple con los criterios establecidos (por ejemplo,

validaciones de seguridad o configuraciones correctas), en línea con las buenas prácticas de desarrollo continuo y la adopción de metodologías DevSecOps para mantener la calidad y la seguridad del software bancario.

4.5.5 Indicador Seguridad

Reglas Generales.

- Todos los casos que impliquen riesgos de seguridad deben obligatoriamente ser corregidos.
- No debe existir vulnerabilidades en la parte superior del cuadro (Very High, High)
- La Tabla 22 muestra los resultados de la revisión de código realizada, incluyendo el total de líneas inspeccionadas, la cantidad de bugs y vulnerabilidades detectadas, la cifra de code smells, así como los porcentajes de cobertura de pruebas y de duplicados en el proyecto. Estos indicadores proporcionan una visión integral de la calidad y el estado del código fuente tras aplicar prácticas de desarrollo y validación continua.

Tabla 22
Métricas de Revisión de Código

No	Descripción	Valor
1	Líneas de Código revisadas.	1245
2	Total Bugs	0
3	Total Vulnerabilidades	0
4	Total Code Smells	35
5	Porcentaje Coberturas	67
6	Porcentaje Duplicados	0

Los datos evidencian que se revisaron 1.245 líneas de código, sin que se hallaran bugs ni vulnerabilidades. Sin embargo, se reportan 35 code smells, lo que sugiere la necesidad de refinar ciertos aspectos de diseño o legibilidad, aun cuando no constituyan fallas de seguridad. El 67 % de cobertura de pruebas refleja un nivel moderado de verificación automatizada, mientras que el 0 % de duplicados indica que no existen porciones de código redundantes. En conjunto, estas métricas revelan un estado sólido en cuanto a seguridad y estabilidad, aunque con oportunidades de mejora para optimizar la calidad y el mantenimiento del código.

4.5.6 Línea Base

La Figura 143 exhibe el resultado del análisis de código realizado en SonarQube, donde se muestra el estado de la Quality Gate, así como indicadores relativos a la seguridad, fiabilidad, mantenibilidad y cobertura de pruebas. Esta vista ofrece un resumen de la calidad global del proyecto, permitiendo a los equipos de desarrollo y seguridad identificar rápidamente las áreas que requieren atención.

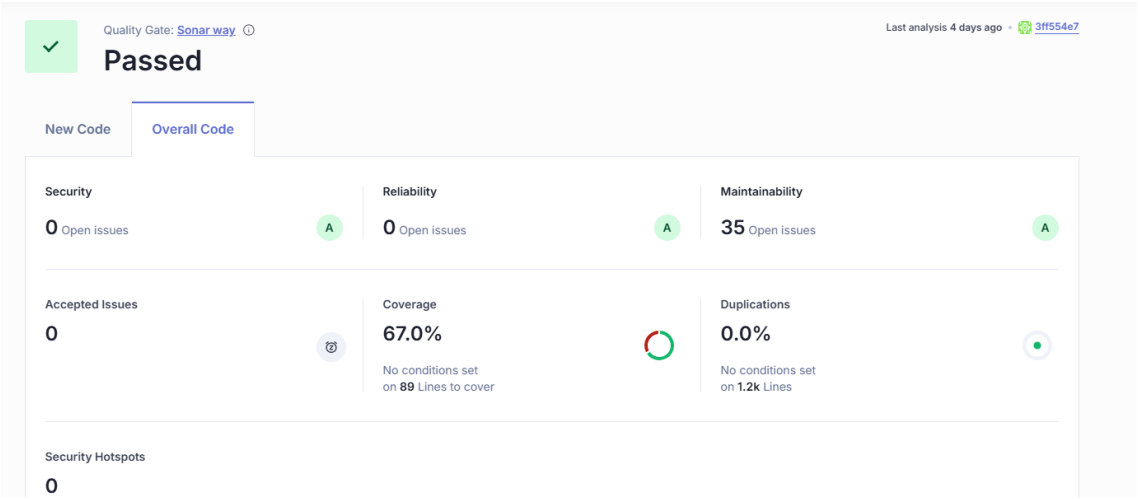


Figura 142. Resultados de Análisis en SonarQube (Quality Gate)

En la captura se observa que la evaluación ha pasado la Quality Gate, con cero incidencias de seguridad y un nivel aceptable de fiabilidad y mantenibilidad. Si bien se registran 35 code smells, éstos no afectan directamente la seguridad, sino que representan oportunidades de refactorización para mejorar la legibilidad y la robustez del código. La cobertura de pruebas alcanza un 67 %, indicando un grado de verificación automatizada moderado, mientras que la duplicación de líneas de código permanece en 0 %, lo que sugiere una buena práctica de diseño y una reducción de riesgos asociados a la repetición de lógicas. Estos indicadores demuestran un estado sólido del proyecto, alineado con las buenas prácticas de un enfoque DevSecOps, en el que la detección y remediación de vulnerabilidades se integran de forma temprana y continua en el ciclo de vida del desarrollo.

4.5.7 Entrega

La Figura144 muestra el resultado del análisis de SonarQube sobre la rama principal (master) del proyecto, con énfasis en la sección de "Nuevo Código". Este panel brinda información detallada acerca de la calidad y la seguridad del código introducido recientemente, reflejando indicadores cruciales como la cobertura de pruebas, la duplicación de líneas y los llamados "Security Hotspots".

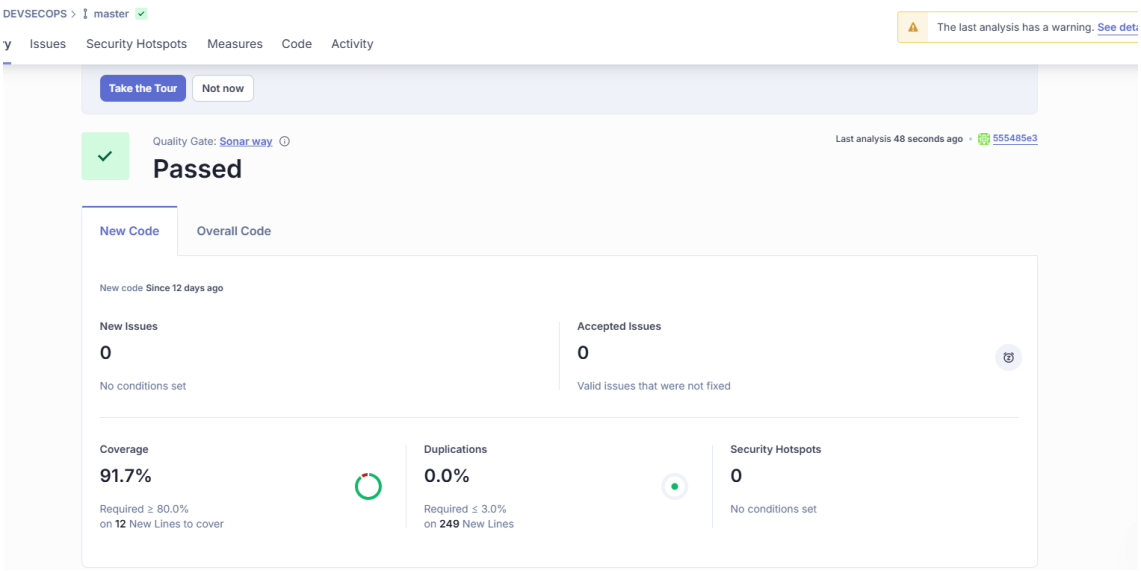


Figura 143. Resultados de Análisis en SonarQube (Nuevo Código)

En la captura se observa que la evaluación ha pasado satisfactoriamente la Quality Gate, sin que se hayan detectado issues de seguridad ni incidencias asociadas a vulnerabilidades. El porcentaje de cobertura de pruebas se sitúa en 91.79 %, lo que representa un nivel muy alto de verificación automática sobre el nuevo código agregado al repositorio. Asimismo, la duplicación de líneas permanece en 0.0 %, indicando la inexistencia de bloques de código repetidos, mientras que los "Security Hotspots" están en cero, reforzando la solidez de la implementación. Estos resultados confirman la efectividad de las prácticas DevSecOps, al integrar la revisión de calidad y seguridad en cada commit, y sugieren un desarrollo encaminado a reducir la presencia de defectos y vulnerabilidades en la aplicación.

4.5.8 Comparativo de revisión de código Global

La siguiente tabla presenta la evolución de diversos indicadores de calidad y seguridad del código, comparando los valores obtenidos en la Línea Base con los resultados de la Entrega final. Estos datos permiten evaluar el impacto de las acciones correctivas y la adopción de prácticas DevSecOps a lo largo del ciclo de desarrollo.

Tabla 23
Comparativa de Métricas entre Línea Base y Entrega

No	Descripción	Línea Base	Entrega
1	Líneas de Código revisadas.	1245	1268
2	Total Bugs	0	0
3	Total Vulnerabilidades	0	0
4	Total Code Smells	35	0
5	Porcentaje Coberturas	67	91.7
6	Porcentaje Duplicados	0	0

Los resultados evidencian un incremento en las líneas de código revisadas (de 1.245 a 1.268), manteniendo cero bugs y vulnerabilidades en ambos puntos de referencia. Llama la atención la eliminación completa de code smells (de 35 a 0), lo que refleja mejoras sustanciales en la legibilidad y mantenibilidad del proyecto. Por otro lado, la cobertura de pruebas aumenta de un 67 % a un 91,7 %, indicando un nivel más profundo de verificación automatizada. Finalmente, el 0 % de duplicados se mantiene inalterado, confirmando la ausencia de segmentos de código redundantes y reforzando la eficacia de las prácticas de revisión y refactorización adoptadas.

4.5.8.1. Indicador Globa Coveragel – Línea Base

La Figura 146 muestra el reporte de cobertura de código de varios archivos que conforman el proyecto WebSocketServer. En esta vista, se detalla el porcentaje de líneas y condiciones cubiertas por las pruebas automatizadas para cada clase principal (por ejemplo, DemoApplication.java, WebSocketService.java y NotificationService.java), así como la existencia de dos componentes adicionales con una cobertura del 100 %.

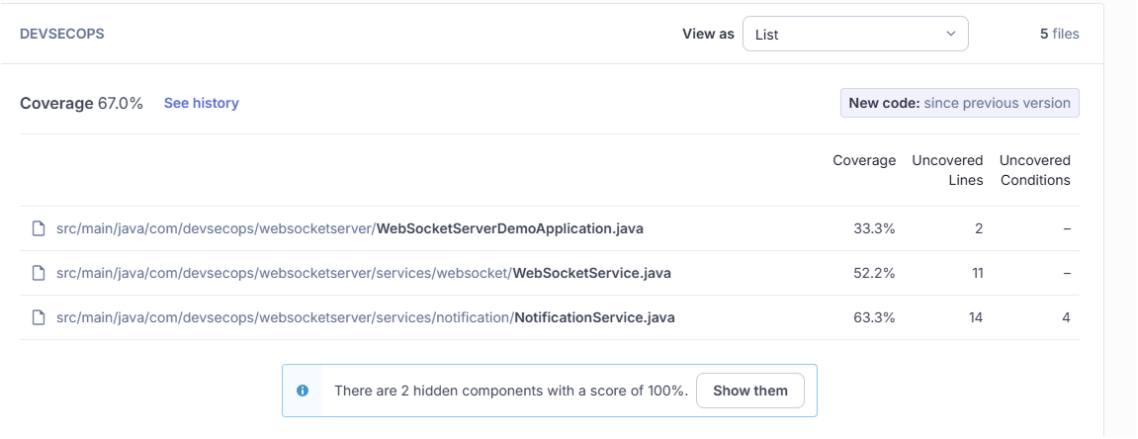


Figura 144. Cobertura de Código en Archivos del Proyecto WebSocketServer

Regla General.

- El indicador global de la entrega debe ser mayor o igual al 90%.

La cobertura global se sitúa en un 67.0 %, distribuyéndose de forma heterogénea entre los diferentes archivos. Por ejemplo, DemoApplication.java alcanza un 33.3 % de cobertura, mientras que WebSocketService.java y NotificationService.java registran valores de 52.2 % y 63.3 %, respectivamente. Esta disparidad indica que, si bien existen componentes con un nivel adecuado de validación, aún hay oportunidades de incrementar el alcance de las pruebas unitarias o de integración para reforzar la confiabilidad y seguridad del sistema. El hecho de que algunos archivos presenten una cobertura total (100 %) sugiere la eficacia de ciertos test cases, sirviendo como referencia para replicar buenas prácticas de verificación y consolidar así el enfoque DevSecOps en la gestión de calidad del software bancario.

4.5.8.2. Indicador Global Coverage – Entrega.

La Figura 147 muestra el nivel de cobertura de pruebas para el nuevo código incorporado en el proyecto WebSocketServer. En esta sección se listan únicamente los archivos modificados o añadidos recientemente, indicando la proporción de líneas y condiciones que han sido validadas mediante pruebas automatizadas.

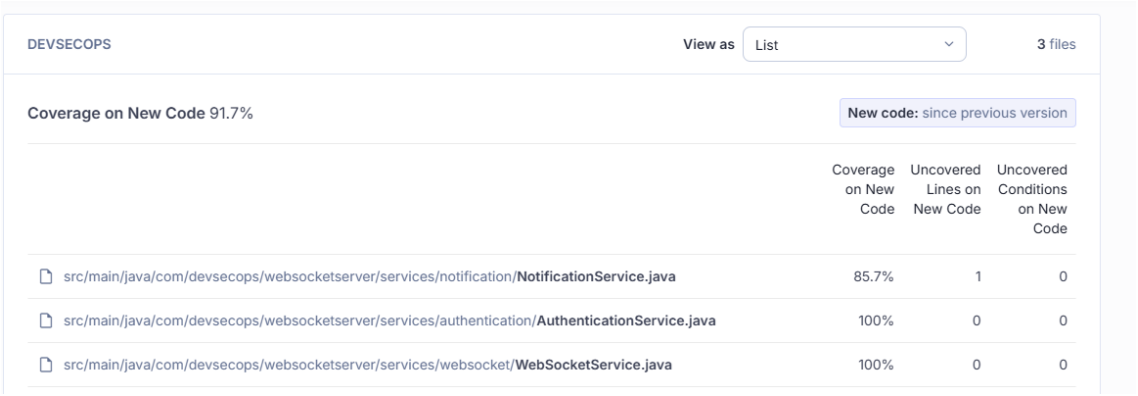


Figura 145. Cobertura de Código en Nuevos Archivos del Proyecto WebSocketServer

Regla General.

- El indicador global de la entrega debe ser mayor o igual al 90%.

Los resultados evidencian una cobertura promedio del 91.7 % en el nuevo código, con archivos como AuthenticationService.java y WebSocketService.java alcanzando el 100 % de verificación. Mientras tanto, NotificationService.java presenta un 85.7 %, lo cual continúa siendo un valor elevado, si bien podría optimizarse para igualar el desempeño de los otros componentes. Esta tendencia refuerza la eficacia de la estrategia DevSecOps, ya que la incorporación de pruebas en etapas tempranas y continuas del desarrollo permite detectar y corregir fallas con rapidez, mejorando la calidad y confiabilidad del software bancario a lo largo de todo el ciclo de vida.

4.5.9 Resumen de Corrección de Defectos

La siguiente tabla presenta un panorama de los principales hallazgos detectados y las acciones de corrección aplicadas en la aplicación, contemplando aspectos de fiabilidad (bugs), seguridad (vulnerabilidades), mantenibilidad (code smells) y duplicación de código. Además, se incluye un análisis del riesgo de no abordar cada uno de estos hallazgos, teniendo en cuenta la complejidad y el alcance del core bancario y sus integraciones con servicios externos.

Tabla 24.
Resumen de Hallazgos y Correcciones en la Aplicación

No	Característica	Encontrados	Corregidos	Riesgo de no corregir
1	Fiabilidad (Bugs)	0	0	Bloqueante / Pueden causar fallos inesperados en la aplicación, pérdida de datos, interrupciones del servicio y una mala experiencia del usuario
2	Seguridad (Vulnerabilidades)	0	0	Bloqueante / Pueden ser explotadas por atacantes para obtener acceso no autorizado a datos confidenciales, comprometer sistemas o realizar ataques maliciosos
3	Mantenibilidad (Code Smells)	35	35	Alto / Dificultan la comprensión, modificación y actualización del código.
4	Duplicación de código	0	0	Medio / Un error en el código duplicado, debe corregirse en múltiples lugares, lo que aumenta el riesgo de olvidar alguna instancia.
Total		35	0	

Los datos indican que no se encontraron bugs ni vulnerabilidades en el proyecto, lo que sugiere un estado sólido en términos de fiabilidad y seguridad. Sin embargo, se identificaron 35 code smells relacionados con la mantenibilidad, los cuales fueron corregidos en su totalidad, eliminando posibles puntos de deuda técnica que podrían dificultar el mantenimiento futuro del sistema. La duplicación de código permanece en 0, evidenciando una adecuada organización y refactorización. Aunque el riesgo de no corregir estos hallazgos se describe como “Ninguno / No aplica”, su atención contribuye a reforzar la calidad global de la aplicación, asegurando que el microservicio mantenga un estándar alto de confiabilidad y manteniendo la complejidad del core bancario bajo control.

4.5.10 Informe Final:

La siguiente tabla compila un total de 35 hallazgos de mantenibilidad, clasificados según su severidad (desde “Menor” o “Informativo” hasta “Crítico” o “Bloqueante”) y con un estado final de “Resuelto” o “Justificado”. Estos hallazgos abarcan problemas frecuentes de legibilidad y coherencia del código, como la duplicación de literales de cadena, las importaciones innecesarias, la visibilidad inadecuada de clases y métodos de prueba (JUnit5), el uso de excepciones genéricas, y la falta de cumplimiento de ciertas reglas de serialización (campos no transitorios). Asimismo, se incluyen casos relacionados con la convención de nomenclatura, la eliminación de variables no utilizadas, y la sustitución de

tipos de datos como java.lang.Boolean por un manejo más seguro en caso de valores nulos. En conjunto, estas incidencias revelan que la mayoría de las deficiencias se concentran en la mejora de la legibilidad y la adaptabilidad del código, lo cual no afecta directamente la seguridad, pero sí la mantenibilidad y la eficiencia en futuros desarrollos. El hecho de que cada problema esté resuelto o justificado demuestra la adopción de un enfoque de mejora continua, alineado con las prácticas DevSecOps, donde la calidad del código se integra de manera temprana y sistemática a lo largo del ciclo de vida del desarrollo.

Tabla 25
Hallazgos de Mantenibilidad y su Resolución en el Proyecto

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
1	webSocketServer/pom.xml	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Bloqueante	Mantenibilidad	141	Resuelta	El código comentado distrae la atención del código ejecutado actualmente. Crea un ruido que aumenta el código de mantenimiento. Y como nunca se ejecuta, rápidamente queda obsoleto y no válido.
2	src/.../websocketserver/config/RedisClusterConfig.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Menor	Mantenibilidad	9	justificado	Las importaciones innecesarias se refieren a la importación de tipos que no se utilizan ni se mencionan en ninguna parte del código. Aunque no afectan el comportamiento de ejecución de la aplicación después de la compilación, eliminarlos hará lo siguiente: Mejorar la legibilidad y mantenibilidad del código. Ayude a evitar posibles conflictos de nombres. Mejora el tiempo de compilación, ya que el compilador tiene menos líneas para leer y menos tipos para resolver. Reduce la cantidad de elementos que el editor de código mostrará para completar automáticamente, mostrando así menos sugerencias irrelevantes. Excepciones Se ignoran las importaciones de los tipos mencionados en Javadocs.
3	src/.../websocketserver/config/RedisClusterConfig.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Menor	Mantenibilidad	45	justificado	Cuando se utiliza el tipo de cuadro java.lang.Boolean como expresión para determinar el flujo de control (como se describe en la Especificación del lenguaje Java §4.2.5 El tipo booleano y los valores booleanos), se generará una excepción NullPointerException si el valor es nulo (como se define en la Especificación del lenguaje Java §5.1.8 Conversión de unboxing). Es más seguro evitar dicha conversión por completo y manejar el valor nulo explícitamente.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Sin embargo, tenga en cuenta que no se plantearán problemas para los booleanos que ya hayan sido verificados como nulos o que estén marcados como @NonNull/@NotNull. Ejemplo de código no compatible Booleano b = getBoolean();if (b) { // No cumple, arrojará NPE cuando b == null foo();} demás { bar();}
4	src/.../websocketserver/config/RedisConfig.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Menor	Mantenibilidad	7	Resuelto	Las importaciones innecesarias se refieren a la importación de tipos que no se utilizan ni se mencionan en ninguna parte del código. Aunque no afectan el comportamiento de ejecución de la aplicación después de la compilación, eliminarlos hará lo siguiente: Mejorar la legibilidad y mantenibilidad del código. Ayude a evitar posibles conflictos de nombres. Mejora el tiempo de compilación, ya que el compilador tiene menos líneas para leer y menos tipos para resolver. Reduce la cantidad de elementos que el editor de código mostrará para completar automáticamente, mostrando así menos sugerencias irrelevantes. Excepciones Se ignoran las importaciones de los tipos mencionados en Javadocs.
5	src/.../websocketserver/config/RedisConfig.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Menor	Mantenibilidad	8	Resuelto	Las importaciones innecesarias se refieren a la importación de tipos que no se utilizan ni se mencionan en ninguna parte del código.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Aunque no afectan el comportamiento de ejecución de la aplicación después de la compilación, eliminarlos hará lo siguiente: Mejora la legibilidad y mantenibilidad del código. Ayude a evitar posibles conflictos de nombres. Mejora el tiempo de compilación, ya que el compilador tiene menos líneas para leer y menos tipos para resolver. Reduce la cantidad de elementos que el editor de código mostrará para completar automáticamente, mostrando así menos sugerencias irrelevantes. Excepciones Se ignoran las importaciones de los tipos mencionados en Javadocs.
6	src/main/java/com/devsecops/websocketserver/entities/generic/OutputEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Menor	Mantenibilidad	32	Resuelto	Una variable local no utilizada es una variable que ha sido declarada pero que no se usa en ninguna parte del bloque de código donde está definida. Es un código muerto, que contribuye a una complejidad innecesaria y genera confusión al leer el código. Por lo tanto, debe eliminarse de su código para mantener la claridad y la eficiencia. ¿Cuál es el impacto potencial? Tener variables locales no utilizadas en su código puede generar varios problemas: Legibilidad reducida: las variables no utilizadas pueden hacer que su código sea más difícil de leer. Añade líneas adicionales y complejidad, lo que puede distraer la atención de la lógica principal del código.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Malentendido: cuando otros desarrolladores leen su código, pueden preguntarse por qué se declara una variable pero no se utiliza. Esto puede generar confusión y mala interpretación de la intención del código. Potencial de errores: si una variable se declara pero no se utiliza, podría indicar un error o un código incompleto. Por ejemplo, si declaró una variable con la intención de usarla en un cálculo, pero luego olvidó hacerlo, es posible que su programa no funcione como se esperaba. Problemas de mantenimiento: las variables no utilizadas pueden dificultar el mantenimiento del código. Si un programador ve una variable no utilizada, podría pensar que es un error e intentar "arreglar" el código, introduciendo potencialmente nuevos errores. Uso de la memoria: aunque los compiladores modernos son lo suficientemente inteligentes como para ignorar las variables no utilizadas, no todos los compiladores hacen esto. En tales casos, las variables no utilizadas ocupan espacio en la memoria, lo que lleva a un uso ineficiente de los recursos. En resumen, las variables locales no utilizadas pueden hacer que su código sea menos legible, más confuso y más difícil de mantener, y pueden provocar errores o un uso ineficiente de la memoria. Por lo tanto, lo mejor es eliminarlos.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
7	src/.../entities/generic/body/ErrorEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Mayor	Mantenibilidad	35	Resuelto	Un error de código típico conocido como parámetros de función no utilizados se refiere a parámetros declarados en una función pero que no se utilizan en ninguna parte del cuerpo de la función. Si bien esto puede parecer inofensivo a primera vista, puede generar confusión y posibles errores en el código. Sin tener en cuenta los valores pasados a dichos parámetros, el comportamiento de la función será el mismo, pero la intención del programador ya no se expresará claramente. Por lo tanto, se considera una buena práctica eliminar los parámetros de función que no se utilizan. Excepciones La regla no generará problemas con los parámetros no utilizados: Que están anotados con @javax.enterprise.event.Observes En anulaciones y métodos de implementación En los métodos predeterminados de la interfaz En métodos no privados que solo arrojan o que tienen cuerpos vacíos En métodos anotados, a menos que la anotación sea @SuppressWarning("unchecked") o @SuppressWarning("rawtypes"), en cuyo caso la anotación se ignorará En métodos reemplazables (no finales, o no miembros de una clase final, no estáticos, no privados), si el parámetro está documentado con un javadoc adecuado.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
8	src/.../websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Informativo	Mantenibilidad	29	Resuelto	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicas las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
9	src/.../com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Informativo	Mantenibilidad	43	justificado	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
10	src/.../com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Informativo	Mantenibilidad	50	Resuelto	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
11	src/.../com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Informativo	Mantenibilidad	69	Resuelto	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
12	src/.../com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro.	Informativa	Mantenibilidad	80	Resuelto	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
13	src/.../com/devsecops/websocketserver/services/WebSocketServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Informativo	Mantenibilidad	19	Resuelto	JUnit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
14	src/.../com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Informativo	Mantenibilidad	87	Resuelto	Unit5 es más tolerante con respecto a la visibilidad de las clases y métodos de prueba que JUnit4, que requería que todo fuera público. Las clases y métodos de prueba pueden tener cualquier visibilidad excepto la privada. Sin embargo, se recomienda utilizar la visibilidad del paquete predeterminada para mejorar la legibilidad. No es necesario que las clases de prueba, los métodos de prueba y los métodos de ciclo de vida sean públicos, pero no deben ser privados. Generalmente se recomienda omitir el modificador público para las clases de prueba, los métodos de prueba y los métodos de ciclo de vida a menos que exista una razón técnica para hacerlo (por ejemplo, cuando una clase de prueba extiende una clase de prueba en otro paquete). Otra razón técnica para hacer públicos las clases y los métodos es simplificar las pruebas en la ruta del módulo cuando se utiliza Java Module System. Guía del usuario de JUnit5 ¿Cuál es el impacto potencial? El código no será convencional y la legibilidad puede verse ligeramente afectada. Excepciones Esta regla no plantea un problema cuando la visibilidad se establece en privada, porque JUnit5 ignora sistemáticamente los métodos y clases de prueba privados, sin una advertencia adecuada. En este caso, también hay un impacto en la confiabilidad y por eso lo maneja la regla S5810.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

15	src/.../websocketserver/config/RedisConfig.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Menor	Mantenibilidad	47	Resuelto	Cuando se utiliza el tipo de cuadro java.lang.Boolean como expresión para determinar el flujo de control (como se describe en la Especificación del lenguaje Java §4.2.5 El tipo booleano y los valores booleanos), se generará una excepción NullPointerException si el valor es nulo (como se define en la Especificación del lenguaje Java §5.1.8 Conversión de unboxing).
							
							Es más seguro evitar dicha conversión por completo y manejar el valor nulo explícitamente.
							
							Sin embargo, tenga en cuenta que no se plantearán problemas para los booleanos que ya hayan sido verificados como nulos o que estén marcados como @NonNull/@NotNull.
							
							Booleano b = getBoolean();
							if (b) { // No cumple, arrojará NPE cuando b == null
							foo();
							} demás {
							bar();
							}
							Solución compatible
							Booleano b = getBoolean();
							if (Boolean.TRUE.equals(b)) {
							foo();
							} demás {

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							bar(); // será invocado tanto para b == false como para b == null
							
							}
							Booleano b = getBoolean();
							si(b != nulo){
							Prueba de cadena = b? "prueba" : "";
							}
							Excepciones
							El problema no surge si la expresión está anotada como @NonNull/@NotNull. Esto es útil si un tipo encuadrado es una instancia de un parámetro de tipo genérico y no se puede evitar.
							
							Lista<Boolean> lista = nueva ArrayList<>();
							lista.add(verdadero);
							lista.add(falso);
							list.forEach((@valor booleano no nulo) -> {
							// Cumple
							si(valor) {
							System.out.println("sí");
							}
							});
							@NonNull Boolean someMethod() { /* ... */ }

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							// Cumple
							
							if(algún método()) { /* ... */ }
							
							@NonNull Boolean boxedNonNull = Boolean.TRUE;
							
							// Cumple
							
							if(boxedNonNull) { /* ... */ }

16	src/.../entities/authentication/TypeAuthenticationResponseEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	20	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco. Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia. Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase. Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
17	src/main/java/com/devsecops/websocketserver/entities/generic/InputEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	22	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco. Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia. Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase. Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.
18	src/.../entities/generic/OutputEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	23	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia. Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase. Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.
19	src/main/java/com/devsecops/websocketserver/entities/generic/PayloadEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critico	Mantenibilidad	16	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco. Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase. Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.
20	src/main/java/com/devsecops/websocketserver/entities/generic/PayloadEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	17	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco. Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia. Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.
21	src/main/java/com/devsecops/websocketserver/entities/message/MessageResponseEntity.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	22	Resuelto	Los campos no estáticos en una clase Serializable deben ser serializables o transitorios. Incluso si la clase nunca se serializa o deserializa explícitamente, no es seguro asumir que esto no pueda suceder. Por ejemplo, bajo carga, la mayoría de los marcos de aplicaciones J2EE descargan objetos en el disco. Un objeto que implementa Serializable pero contiene miembros de datos no transitorios y no serializables (y por lo tanto viola el contrato) podría causar fallas en la aplicación y abrir la puerta a los atacantes. En general, se espera que una clase Serializable cumpla su contrato y no muestre un comportamiento inesperado cuando se serializa una instancia. Esta norma plantea una cuestión sobre: Campos no serializables. Cuando a un campo se le asigna un tipo no serializable dentro de la clase. Campos de colección cuando no son privados. Los valores que no son serializables se podrían agregar a estas colecciones de forma externa. Debido al borrado de tipos, no se puede garantizar que la colección solo contendrá objetos serializables en tiempo de ejecución a pesar de estar declarada como una colección de tipos serializables.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

22	src/main/java/com/devsecops/websocketserver/services/authentication/AuthenticationService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Menor	Mantenibilidad	19	Resuelto	Una convención de nomenclatura en el desarrollo de software es un conjunto de pautas para nombrar elementos de código como variables, funciones y clases.
							
							El objetivo de una convención de nomenclatura es hacer que el código sea más legible y comprensible, lo que facilita su mantenimiento y depuración. También garantiza la coherencia en el código, especialmente cuando varios desarrolladores trabajan en el mismo proyecto.
							
							Esta regla verifica que los nombres de los campos coincidan con una expresión regular proporcionada.
							
							Usando la expresión regular ^[a-z][a-zA-Z0-9]*$, el siguiente código no conforme:
							
							clase MiClase {
							
							privado int mi_campo;
							
							}
							
							Debe ser reemplazado por:
							
							clase MiClase {
							
							privado en mi campo;
							
							}

23	src/main/java/com/devsecops/websocketserver/services/authentication/IAuthenticationService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Importante	Mantenibilidad	14	Resuelto	Lanzar excepciones genéricas como Error, RuntimeException, Throwable y Exception tendrá un impacto negativo en cualquier código que intente detectar estas excepciones.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Desde la perspectiva del consumidor, generalmente es una buena práctica detectar solo las excepciones que desea manejar. Lo ideal sería permitir que otras excepciones se propaguen por el seguimiento de la pila para que puedan tratarse adecuadamente. Cuando se lanza una excepción genérica, obliga a los consumidores a detectar excepciones que no pretenden manejar, que luego deben volver a lanzar. Además, cuando se trabaja con un tipo genérico de excepción, la única forma de distinguir entre múltiples excepciones es verificar su mensaje, que es propenso a errores y difícil de mantener. Las excepciones legítimas pueden ser silenciadas involuntariamente y los errores pueden ocultarse. Por ejemplo, cuando se detecta un Throwable y no se vuelve a lanzar, puede enmascarar errores como OutOfMemoryError e impedir que el programa finalice correctamente. Por lo tanto, al lanzar una excepción, se recomienda lanzar la excepción más específica posible para que los consumidores puedan manejarla intencionalmente. Excepciones Las excepciones genéricas en las firmas de los métodos de reemplazo se ignoran, porque un método de reemplazo tiene que seguir la firma de la declaración de lanzamiento en la superclase. El problema se planteará en la declaración de superclase del método (o no se planteará en absoluto si la superclase no forma parte del análisis). @Anular public void myMethod() lanza una excepción {...}

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Las excepciones genéricas también se ignoran en las firmas de métodos que realizan llamadas a métodos que generan excepciones genéricas.
							
							public void myOtherMethod() lanza una excepción {
							
							hacerLaCosa(); // este método arroja una excepción
							
							}

24	src/main/java/com/devsecops/websocketserver/services/notification/NotificationService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	29	Resuelto	Los literales de cadena duplicados hacen que el proceso de refactorización sea complejo y propenso a errores, ya que cualquier cambio debería propagarse en todas las apariciones. Excepciones Para evitar generar algunos falsos positivos, se excluyen los literales que tengan menos de 5 caracteres.
25	src/main/java/com/devsecops/websocketserver/services/notification/NotificationService.java	Adaptabilidad Para que sea intencional, el contenido del código debe ser preciso y tener un propósito.	Critica	Mantenibilidad	32	Resuelto	Los literales de cadena duplicados hacen que el proceso de refactorización sea complejo y propenso a errores, ya que cualquier cambio debería propagarse en todas las apariciones. Excepciones Para evitar generar algunos falsos positivos, se excluyen los literales que tengan menos de 5 caracteres.
26	src/main/java/com/devsecops/websocketserver/services/notification/NotificationService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Importante	Mantenibilidad	93	Resuelto	Lanzar excepciones genéricas como Error, RuntimeException, Throwable y Exception tendrá un impacto negativo en cualquier código que intente detectar estas excepciones.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Desde la perspectiva del consumidor, generalmente es una buena práctica detectar solo las excepciones que desea manejar. Lo ideal sería permitir que otras excepciones se propaguen por el seguimiento de la pila para que puedan tratarse adecuadamente. Cuando se lanza una excepción genérica, obliga a los consumidores a detectar excepciones que no pretenden manejar, que luego deben volver a lanzar. Además, cuando se trabaja con un tipo genérico de excepción, la única forma de distinguir entre múltiples excepciones es verificar su mensaje, que es propenso a errores y difícil de mantener. Las excepciones legítimas pueden ser silenciadas involuntariamente y los errores pueden ocultarse. Por ejemplo, cuando se detecta un Throwable y no se vuelve a lanzar, puede enmascarar errores como OutOfMemoryError e impedir que el programa finalice correctamente. Por lo tanto, al lanzar una excepción, se recomienda lanzar la excepción más específica posible para que los consumidores puedan manejarla intencionalmente. Excepciones Las excepciones genéricas en las firmas de los métodos de reemplazo se ignoran, porque un método de reemplazo tiene que seguir la firma de la declaración de lanzamiento en la superclase. El problema se planteará en la declaración de superclase del método (o no se planteará en absoluto si la superclase no forma parte del análisis). @Anular public void myMethod() lanza una excepción {...}

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Las excepciones genéricas también se ignoran en las firmas de métodos que realizan llamadas a métodos que generan excepciones genéricas.
							
							public void myOtherMethod() lanza una excepción {
							
							hacerLaCosa(); // este método arroja una excepción
							
							}

27	src/main/java/com/devsecops/websocketserver/services/websocket/IWebSocketService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Importante	Mantenibilidad	14	Resuelto	Lanzar excepciones genéricas como Error, RuntimeException, Throwable y Exception tendrá un impacto negativo en cualquier código que intente detectar estas excepciones. Desde la perspectiva del consumidor, generalmente es una buena práctica detectar solo las excepciones que desea manejar. Lo ideal sería permitir que otras excepciones se propaguen por el seguimiento de la pila para que puedan tratarse adecuadamente. Cuando se lanza una excepción genérica, obliga a los consumidores a detectar excepciones que no pretenden manejar, que luego deben volver a lanzar. Además, cuando se trabaja con un tipo genérico de excepción, la única forma de distinguir entre múltiples excepciones es verificar su mensaje, que es propenso a errores y difícil de mantener. Las excepciones legítimas pueden ser silenciadas involuntariamente y los errores pueden ocultarse. Por ejemplo, cuando se detecta un Throwable y no se vuelve a lanzar, puede enmascarar errores como OutOfMemoryError e impedir que el programa finalice correctamente. Por lo tanto, al lanzar una excepción, se recomienda lanzar la excepción más específica posible para que los consumidores puedan manejarla intencionalmente. Excepciones

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Las excepciones genéricas en las firmas de los métodos de reemplazo se ignoran, porque un método de reemplazo tiene que seguir la firma de la declaración de lanzamiento en la superclase. El problema se planteará en la declaración de superclase del método (o no se planteará en absoluto si la superclase no forma parte del análisis).
							
							@Anular
							
							public void myMethod() lanza una excepción {...}
							
							Las excepciones genéricas también se ignoran en las firmas de métodos que realizan llamadas a métodos que generan excepciones genéricas.
							
							public void myOtherMethod() lanza una excepción {
							
							hacerLaCosa(); // este método arroja una excepción
							
							}

28	src/test/java/com/devsecops/websocketserver/WebSocketServerDemoApplicationTests.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Critica	Mantenibilidad	10	Resuelto	Un método vacío generalmente se considera una mala práctica y puede generar confusión, problemas de legibilidad y mantenimiento. Los métodos vacíos no aportan ninguna funcionalidad y son engañosos para los demás, ya que podrían pensar que la implementación del método cumple con un requisito específico e identificado. Hay varias razones por las que un método no tiene cuerpo: Es una omisión involuntaria y debe corregirse para evitar un comportamiento inesperado en la producción. Todavía no cuenta con apoyo, o nunca lo tendrá. En este caso se debe lanzar una excepción.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							El método es una anulación intencional en blanco. En este caso, un comentario anidado debería explicar el motivo de la anulación del espacio en blanco.
							
							Excepciones
							
							Esto no plantea ningún problema en los siguientes casos:
							
							Constructores predeterminados no públicos (sin argumentos)
							
							Constructores públicos predeterminados (sin argumentos) cuando hay otros constructores en la clase
							
							Métodos vacíos en clases abstractas.
							
							Métodos anotados con @org.aspectj.lang.annotation.Pointcut()
							
							clase abstracta pública Animal {
							void talk() { // implementación predeterminada ignorada
							}
							}

29	src/test/java/com/devsecops/websocketserver/services/AuthenticationServiceTest.java	Esta es una cuestión de coherencia. Para ser coherente, el código debe escribirse de forma uniforme y a convencional.	Menor	Mantenibilidad	31	Resuelto	Una convención de nomenclatura en el desarrollo de software es un conjunto de pautas para nombrar elementos de código como variables, funciones y clases. El objetivo de una convención de nomenclatura es hacer que el código sea más legible y comprensible, lo que facilita su mantenimiento y depuración. También garantiza la coherencia en el código, especialmente cuando varios desarrolladores trabajan en el mismo proyecto.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Esta regla verifica que los nombres de los campos coincidan con una expresión regular proporcionada.
							
							Usando la expresión regular ^[a-z][a-zA-Z0-9]*$, el siguiente código no conforme:
							
							clase MiClase {
							
							privado int mi_campo;
							
							}
							
							Debe ser reemplazado por:
							
							clase MiClase {
							
							privado en mi campo;
							
							}

30	src/test/java/com/devsecops/websocketserver/services/NotificationServiceTest.java	Esta es una cuestión de coherencia. Para ser coherente, el código debe escribirse de forma uniforme y convencional.	Menor	Mantenibilidad	40	Resuelto	Las constantes son variables cuyo valor no cambia durante el tiempo de ejecución de un programa después de la inicialización. A menudo, las constantes se utilizan en varias ubicaciones en diferentes subrutinas. Es importante que los nombres de las constantes sigan un patrón coherente y fácilmente reconocible. De esta manera, los lectores comprenden inmediatamente que el valor al que se hace referencia no cambia, lo que simplifica la depuración. O, en el caso de constantes primitivas, que el acceso a la constante sea seguro para subprocesos. Esta regla verifica que todos los nombres de constantes coincidan con una expresión regular determinada. ¿Cuál es el impacto potencial?

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación
							Ignorar la convención de nomenclatura para las constantes hace que el código sea menos legible ya que las constantes y las variables son más difíciles de distinguir. El código que es difícil de entender también lo es de mantener entre los diferentes miembros del equipo.
31	src/test/java/com/devsecops/websocketserver/services/WebSocketServiceTest.java	Esta es una cuestión de adaptabilidad. Para ser adaptable, el código debe estar estructurado para que sea fácil de evolucionar y desarrollar con confianza.	Bloqueador	Mantenibilidad	77	Resuelto	Un caso de prueba sin aserciones sólo garantiza que no se produzcan excepciones. Más allá de la ejecutabilidad básica, no garantiza nada sobre el comportamiento del código bajo prueba.
32	src/main/java/com/devsecops/websocketserver/services/notification/NotificationService.java	Esta es una cuestión de adaptabilidad. Para ser adaptable, el código debe estar estructurado para que sea fácil de evolucionar y desarrollar con confianza.	Critico	Mantenibilidad	75	Resuelto	Los literales de cadena duplicados hacen que el proceso de refactorización sea complejo y propenso a errores, ya que cualquier cambio debería propagarse en todas las apariciones. Excepciones Para evitar generar algunos falsos positivos, se excluyen los literales que tengan menos de 5 caracteres.
33	src/main/java/com/devsecops/websocketserver/services/notification/NotificationService.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Informativo	Mantenibilidad	89	Resuelto	En Java 14 hay una nueva forma de escribir casos en Switch Statement y Expression cuando se debe realizar la misma acción para diferentes casos. En lugar de declarar varias ramas con la misma acción, puede combinarlas todas en un solo grupo de casos, separadas por comas. Dará como resultado un código más conciso y una legibilidad mejorada.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Esta regla informa un problema cuando se pueden agrupar varios casos en un Switch en un solo caso separado por comas.
							
							Ejemplo de código no compatible
							// Cambiar expresión
							int i = cambiar (modo) {
							caso "a":
							caso "b":
							rendimiento 1;
							por defecto:
							rendimiento 3;
							};
							// Declaración de cambio
							cambiar (modo) {
							caso "a":
							caso "b":
							hacer algo();
							romper;
							por defecto:
							hacer algo más();
							}

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

Solución compatible

```
// Cambiar expresión
int i = cambiar (modo) {
  caso "a", "b":
    rendimiento 1;
  por defecto:
    rendimiento 3;
};
// Declaración de cambio
cambiar (modo) {
  caso "a", "b":
    hacer algo();
    romper;
  por defecto:
    hacer algo más();
}
// O incluso mejor:
cambiar (modo) {
  caso "a", "b" -> hacerAlgo();
  predeterminado -> hacerAlgoMás();
}
```

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

34	src/test/java/com/devsecops/websocketserver/services/AuthenticationServiceTest.java	Esta es una cuestión de intencionalidad, el código no es lo suficientemente claro	Menor	Mantenibilidad	58	Resuelto	Proporciona comparadores de argumentos para detectar o verificar llamadas a métodos de manera flexible.
							
							Mockito.verify(), Mockito.when(), Stubber.when() y BDDMockito.given() tienen sobrecargas con y sin comparadores de argumentos.
							
							Sin embargo, el comportamiento de coincidencia predeterminado (es decir, sin comparadores de argumentos) utiliza iguales (). Si solo se usa el comparador org.mockito.ArgumentMatchers.eq(), la llamada es equivalente a la llamada sin comparadores, es decir, el eq() no es necesario y se puede omitir. El código resultante es más corto y más fácil de leer.
							
							Ejemplo de código no compatible
							
							@Prueba
							
							public void miPrueba() {
							
							dado(foo.bar(eq(v1), eq(v2), eq(v3))).willReturn(null); // No conforme
							
							cuando(foo.baz(eq(v4), eq(v5))).thenReturn("foo"); // No conforme
							
							doThrow(new RuntimeException()).when(foo).quux(eq(42)); // No conforme
							
							verificar(foo).bar(eq(v1), eq(v2), eq(v3)); // No conforme
							
							}
							
							Solución compatible

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							@Prueba
							public void miPrueba() {
							dado(foo.bar(v1, v2, v3)).willReturn(null);
							cuando(foo.baz(v4, v5)).entoncesReturn("foo");
							doThrow(new RuntimeException()).when(foo).quux(42);
							verificar(foo).bar(v1, v2, v3);
							}

35	src/main/java/com/devsecops/websocketserver/config/RedisClusterConfig.java	Gravedad del impacto. La gravedad ahora está directamente relacionada con la calidad del software afectada. Esto significa que una calidad de software afectada tiene una gravedad. Hay cinco niveles de gravedad: bloqueador, alto, medio, bajo e información.	Mayor	Confiabilidad	44	Resuelto	Las comparaciones de tipos diferentes siempre arrojarán falso. La comparación y todo su código dependiente se pueden eliminar simplemente. Esto incluye: comparando un objeto con nulo comparar un objeto con una primitiva no relacionada (por ejemplo, una cadena con un int) comparar clases no relacionadas comparar una clase y una interfaz no relacionadas comparar tipos de interfaz no relacionados comparar una matriz con una que no es una matriz comparando dos matrices Específicamente en el caso de las matrices, dado que las matrices no anulan Object.equals(), llamar a iguales en dos matrices es lo mismo que comparar sus direcciones. Esto significa que array1.equals(array2) es equivalente a array1==array2.

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							Sin embargo, algunos desarrolladores podrían esperar que Array.equals(Object obj) haga más que una simple comparación de direcciones de memoria, comparando, por ejemplo, el tamaño y el contenido de las dos matrices. En cambio, el operador == o Arrays.equals(array1, array2) siempre debe usarse con matrices.
							
							Ejemplo de código no compatible
							
							interface KitchenTool { ... };
							
							interface Plant {...}
							
							public class Spatula implements KitchenTool { ... }
							
							public class Tree implements Plant { ...}
							
							//...
							
							Spatula spatula = new Spatula();
							
							KitchenTool tool = spatula;
							
							KitchenTool [] tools = {tool};
							
							Tree tree = new Tree();
							
							Plant plant = tree;
							
							Tree [] trees = {tree};
							
							if (spatula.equals(tree)) { // Noncompliant; unrelated classes
							
							// ...
							
							}

No	Fuente	Regla	Prioridad	Característica	Línea	Estado	Observación

							else if (spatula.equals(plant)) { // Noncompliant; unrelated class and interface
							// ...
							}
							else if (tool.equals(plant)) { // Noncompliant; unrelated interfaces
							// ...
							}
							else if (tool.equals(tools)) { // Noncompliant; array & non-array
							// ...
							}
							else if (trees.equals(tools)) { // Noncompliant; incompatible arrays
							// ...
							}
							else if (tree.equals(null)) { // Noncompliant
							// ...
							}

4.5.11 Escaneo de Imágenes con Trivy

Luego de las actualizaciones realizadas en los componentes se logró solventar el 100% de las vulnerabilidades detectadas por la herramienta.

La siguiente figura muestra la sección final del resultado de escaneo generado por Trivy para el contenedor, en la que se incluye una notificación VEX (Vulnerability Exploitability eXchange). Este aviso sugiere que, si el usuario determina que las vulnerabilidades reportadas no son realmente explotables, puede emitir un statement VEX que proporcione transparencia adicional respecto al estado real de seguridad de la imagen.

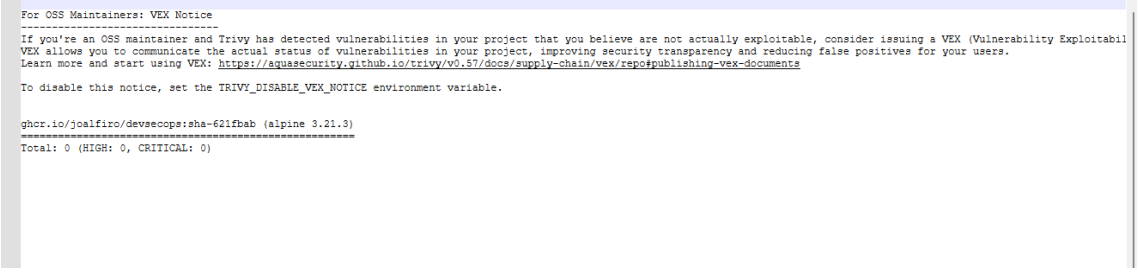


Figura 146 Trivy Output con VEX Notice y Resumen de Vulnerabilidades

En la salida se observa que Trivy no ha identificado vulnerabilidades clasificadas como HIGH o CRITICAL, lo que indica un estado favorable de la imagen evaluada. El VEX Notice brinda la posibilidad de comunicar formalmente cualquier condición en la que las supuestas fallas no representan un riesgo real para el proyecto, reforzando la política de transparencia y seguridad en un entorno DevSecOps. Esta práctica resulta de gran utilidad para minimizar falsos positivos y mantener un registro claro sobre la verdadera criticidad de las incidencias, garantizando la eficacia de las medidas de protección adoptadas.

4.6 Resultados de la implementación de la Fase5: Plan de integración para el desarrollo, seguridad y operaciones

En la Fase 5 se llevó a cabo el diseño completo del flujo de trabajo DevSecOps, abarcando los procesos de desarrollo, seguridad y operaciones en un entorno de pruebas, con el fin de ejecutar experimentos empleando herramientas open source y analizar sus resultados. A partir de esta experiencia, se definió el plan de integración que articula los

procedimientos de desarrollo, seguridad y operaciones de manera coherente. Finalmente, se consolidaron los hallazgos de la investigación en un informe detallado, el cual describe el estudio de forma integral, presenta los resultados obtenidos, las conclusiones alcanzadas y las implicaciones prácticas y teóricas. Dicho informe será compartido con la comunidad académica y profesional para contribuir al avance del conocimiento en el ámbito de DevSecOps y la seguridad del software en entornos de microservicios bancarios. A continuación, se exponen los resultados de esta fase.

Este plan tiene la finalidad de dar a conocer como realizar la integración para el desarrollo seguridad y operaciones una vez instaladas las herramientas seleccionadas en la fase 1.

4.6.1 Desarrollo

4.6.1.1. CI/CD Pipeline con GitHub Actions

Este pipeline implementa un flujo completo de CI/CD (Integración Continua/Entrega Continua) utilizando GitHub Actions, con análisis de seguridad, construcción de imágenes Docker y despliegue automático en AWS EC2.

4.6.1.2. Características

- Análisis de seguridad con Trivy
- Construcción y publicación de imágenes Docker
- Despliegue automático en AWS EC2
- Publicación en GitHub Container Registry (GHCR)
- Integración continua en rama develop

4.6.1.3. Prerrequisitos

- Repositorio GitHub con acceso a GitHub Actions
- Cuenta AWS y una instancia EC2 configurada
- Secrets configurados en el repositorio de GitHub

4.6.1.4. Secrets Necesarios

```
AWS_ACCESS_KEY_ID          # ID de acceso de AWS
AWS_SECRET_ACCESS_KEY      # Clave secreta de AWS
AWS_REGION                 # Región de AWS
EC2_HOST                   # IP o DNS de la instancia EC2
EC2_USERNAME               # Usuario SSH de EC2
EC2_SSH_KEY               # Clave SSH privada para acceso a EC2
GITHUB_TOKEN              # Token de GitHub (automáticamente proporcionado)
```

Figura 147. Variables de Entorno para la Configuración de AWS y GitHub

En la lista se identifican claves de acceso y parámetros necesarios para interactuar con los servicios de AWS (por ejemplo, AWS_ACCESS_KEY_ID y AWS_SECRET_ACCESS_KEY), así como la configuración de la región (AWS_REGION) y la instancia (EC2_HOST, EC2_USERNAME, EC2_SSH_KEY). Además, se incluye el GITHUB_TOKEN, el cual permite a los flujos de trabajo en GitHub Actions autenticar acciones de despliegue o escaneo de código sin exponer las credenciales de manera directa. El uso de variables de entorno no solo aporta un nivel de abstracción y seguridad (al evitar almacenar credenciales en archivos de configuración), sino que también facilita la escalabilidad y la reproducibilidad del pipeline DevSecOps en diferentes entornos y proyectos.

4.6.1.5. Estructura del Pipeline

1. Job: build-and-push

Este job se encarga de la construcción y análisis de seguridad:

- Checkout del código
- Escaneo de vulnerabilidades del código fuente con Trivy
- Login en GitHub Container Registry
- Construcción de la imagen Docker
- Escaneo de vulnerabilidades de la imagen Docker
- Push de la imagen al registro

2. Job: deploy

Este job maneja el despliegue:

- Configuración de credenciales AWS
- Conexión SSH a la instancia EC2
- Despliegue del contenedor Docker

4.6.1.6. Formato de Tags

La Figura 150 muestra el esquema de etiquetado utilizado para subir y versionar imágenes de contenedor en el GitHub Container Registry (GHCR). Este formato de etiqueta incluye tanto el propietario y el repositorio como un identificador único basado en el hash del commit (SHA), lo que permite rastrear con precisión la fuente y el historial de cambios de la imagen.

```
ghcr.io/<owner>/<repository>:sha-<commit-sha>
```

Figura 148. Formato de Etiquetado de Imágenes en GitHub Container Registry

4.6.1.7. Análisis de Seguridad

El pipeline incluye dos niveles de análisis con Trivy:

1. Análisis del Filesystem

- Escanea el código fuente
- Guarda resultados en trivy-results.txt

2. Análisis de la Imagen Docker

- Escanea la imagen construida
- Guarda resultados en trivy-image-results.txt

Los resultados se guardan como artefactos de la acción y están disponibles por 7 días.

4.6.1.8. Configuración del Contenedor

El contenedor se despliega con las siguientes configuraciones:

```
- Nombre: websocket-server
- Puerto: 8081:8081
- Política de reinicio: unless-stopped
```

Figura 149. Configuración del Contenedor Docker para WebSocket-Server

En esta configuración, el contenedor recibe el nombre de websocket-server, exponiendo el puerto 8081 de la aplicación hacia el puerto 8081 del host, lo que facilita el acceso externo a las funcionalidades de WebSocket. La política de reinicio "unless-stopped" garantiza que el contenedor se reinicie de manera automática tras un reinicio del sistema o ante fallas, excepto si ha sido detenido de forma manual. Este enfoque contribuye a la disponibilidad continua del servicio, un aspecto esencial en un entorno DevSecOps donde la resiliencia y la mínima interrupción de los servicios resultan primordiales.

4.6.1.9. Uso

1. Asegúrate de tener todos los secrets configurados en tu repositorio
2. Coloca tu Dockerfile en la ruta webSocketServer/Dockerfile
3. Haz push a la rama develop para activar el pipeline

4.6.1.10. Monitoreo

Puedes monitorear el pipeline en:

- GitHub Actions dashboard del repositorio
- Logs del contenedor en EC2 (docker logs websocket-server)
- Artefactos de escaneo en la sección de Actions

4.6.1.11. Troubleshooting

1. Error de autenticación GHCR
    - Verifica el GITHUB_TOKEN y los permisos del repositorio

2. Error de conexión EC2

    - Verifica las credenciales AWS y la clave SSH
    - Asegúrate de que los puertos necesarios estén abiertos

3. Error en el escaneo Trivy

    - Revisa los artefactos generados para más detalles
    - Verifica la configuración de severidad en el workflow

4.6.1.12. Mantenimiento

- Revisa regularmente los resultados del escaneo de seguridad
- Actualiza las versiones de las actions utilizadas
- Monitorea el uso de recursos en EC2

5 CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

El enfoque DevSecOps demostró que la seguridad puede y debe integrarse de manera sistemática desde las primeras fases del ciclo de desarrollo de software bancario. La implementación de herramientas de análisis estático (SAST), dinámico (DAST) y escaneo de contenedores permitió identificar vulnerabilidades con antelación, reduciendo costos de corrección y fortaleciendo la postura de seguridad en cada etapa.

Soluciones como SonarQube, Trivy, GitHub Actions, Grafana, Docker, Apache Maven, GitHub y GitHub Dashboard mostraron un nivel de aceptación significativo entre los equipos encuestados y entrevistados. Su flexibilidad, costo reducido y respaldo comunitario facilitaron la adopción de prácticas seguras en entornos de microservicios, sin impactar negativamente la agilidad requerida por la banca.

A pesar de las ventajas tecnológicas, la adopción efectiva de DevSecOps depende en gran medida de la cultura interna. Los hallazgos cualitativos (entrevistas) revelaron que la colaboración entre desarrollo, seguridad y operaciones mejora la eficacia de las prácticas y reduce la resistencia al cambio. La alineación con las normas (ISO/IEC 27034, PCI DSS) se ve reforzada cuando todos los equipos comparten la responsabilidad de la protección de datos.

La arquitectura de microservicios, potenciada por contenedores, facilita la escalabilidad de los servicios bancarios. No obstante, también incrementa la superficie de ataque. La automatización de pruebas de seguridad, la orquestación con Docker y el monitoreo continuo (Grafana, Prometheus) han sido determinantes para detectar fallas potenciales y responder con rapidez ante incidentes.

La selección del microservicio para la integración del pago de servicios es fundamental para las operaciones críticas del core bancario permitió validar las prácticas DevSecOps en un entorno que maneja datos sensibles y una arquitectura distribuida. Se evidenció la eficacia de integrar pruebas de seguridad automatizadas, el control de versiones y el despliegue continuo, demostrando que los hallazgos son aplicables a contextos bancarios similares.

Varios expertos coincidieron en que la carencia de indicadores claros para medir el impacto financiero de la seguridad dificulta justificar la inversión en DevSecOps. Aun

así, la reducción del tiempo de respuesta ante incidentes y la prevención de fallas críticas evidencian una mejora sustancial en la calidad y confiabilidad del software bancario.

5.2 Recomendaciones

Para formalizar la estrategia DevSecOps, resulta esencial adoptar un plan de implementación gradual, comenzando por definir metas y cronogramas realistas que permitan integrar la seguridad de manera sistemática en cada fase del desarrollo. Asimismo, se recomienda incluir la documentación de procesos y políticas internas, tomando como referencia tanto los estándares de ISO/IEC 27034 como las guías de OWASP, de modo que la organización disponga de lineamientos claros y consistentes. En este proceso, es aconsejable priorizar los microservicios críticos durante el primer ciclo de adopción, para después extender gradualmente las prácticas DevSecOps al resto de la arquitectura, asegurando así un avance ordenado y una mayor efectividad en la protección de los activos más sensibles.

Para fortalecer la capacitación y la cultura de colaboración, es fundamental desarrollar talleres prácticos que aborden temas como análisis estático/dinámico, contenedores y metodologías ágiles, de manera que los equipos adquieran habilidades técnicas y comprendan la relevancia de la seguridad en cada etapa del desarrollo. Asimismo, se recomienda promover la comunicación continua entre desarrollo, operaciones y seguridad, estableciendo reuniones de retroalimentación y revisión de incidencias que permitan intercambiar conocimientos y detectar oportunidades de mejora. Finalmente, resulta estratégico incentivar la certificación de los equipos en DevOps/DevSecOps y su participación en comunidades técnicas (meetups, foros especializados), impulsando la actualización constante y el intercambio de buenas prácticas en el entorno profesional.

Para implementar métricas de seguridad y calidad de software, se recomienda establecer indicadores de eficiencia, como el tiempo medio de detección y corrección de vulnerabilidades o el número de hallazgos críticos, y complementarlos con indicadores de cobertura de pruebas (SAST, DAST, análisis de contenedores). Además, es conveniente emplear dashboards (por ejemplo, Grafana o SonarQube) para visualizar dichas métricas en tiempo real y facilitar la toma de decisiones. De esta manera, la inversión en seguridad puede justificarse (ROI) al cuantificar la reducción de incidentes

y el costo de correcciones tardías evitadas, demostrando así el valor tangible de las prácticas DevSecOps.

Para adoptar herramientas open source con alto respaldo comunitario, resulta esencial mantener actualizadas soluciones de escaneo como Trivy, OWASP ZAP o SonarQube, integrando además los plugins que mejor se alineen con el stack tecnológico bancario. Asimismo, se recomienda evaluar periódicamente nuevas opciones, tomando en cuenta factores como la escalabilidad, la facilidad de uso y la compatibilidad con entornos bancarios sometidos a estrictas regulaciones, a fin de garantizar una protección continua y adaptada a las necesidades del negocio.

Para reforzar la orquestación y el monitoreo en microservicios, es fundamental establecer pipelines de CI/CD que incluyan validaciones de seguridad y despliegues progresivos, así como la posibilidad de rollbacks automáticos ante eventuales fallas. Asimismo, se recomienda incorporar la observabilidad en cada componente (logging, métricas y tracing), enlazándola con alertas y reportes de seguridad para permitir la detección temprana de anomalías. Además, se debe garantizar el cumplimiento de requisitos regulatorios (por ejemplo, segregación de redes y cifrado de datos) dentro de la infraestructura de contenedores, reforzando así la protección de la información y la integridad de los servicios bancarios.

6 Bibliografía

Ahmad, A., Maynard, S. B., y Shanks, G. (2015). A case analysis of information systems and security incident responses. International Journal of Information Management, 35(6), 717–723. https://doi.org/10.1016/j.ijinfomgt.2015.08.001

Aljohani, M., y Alqahtani, S. (2023). A Unified Framework for Automating Software Security

Aqua Security. (2023). Trivy Documentation. https://github.com/aquasecurity/trivy

Balalaie, A., Heydarnoori, A., y Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 33(3), 42–52. https://doi.org/10.1109/MS.2016.64

Banusevicute, A., Jusas, V., y Rojus, A. (2022). The Impact of DevSecOps Culture on Secure Software Development: A Multi-case Study. En 2022 14th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) (pp. 1–6). IEEE. https://doi.org/10.1109/CogInfoCom55600.2022.9958542

Biehl, M., Campara, Z., Kılıç, S. E., y Cito, J. (2021). Continuous security: A DevSecOps case study. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (pp. 58–62). IEEE. https://doi.org/10.1109/MSR52588.2021.00018

Bozic, T., Resch, M., y Janosevic, M. (2021). Building distributed observability with Prometheus and Grafana in DevOps environments. In Proceedings of the 29th ACM International Conference on Advances in Geographic Information Systems, 270–278. https://doi.org/10.1145/3474717.3484217

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., y Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50–57. https://doi.org/10.1145/2890784

Castelluccio, M., Lisa, C., y de Lucia, A. (2019). Enhancing code review process in pull-based development models: an empirical study on GitHub. In Proceedings of the 27th IEEE International Conference on Program Comprehension (ICPC) (pp. 245–250). IEEE. https://doi.org/10.1109/ICPC.2019.00041

Cheng, L., Liu, X., y Yao, D. (2021). Securing microservices with zero trust networks. In 2021 IEEE Symposium on Security and Privacy Workshops (SPW) (pp. 1–7). IEEE. https://doi.org/10.1109/SPW53761.2021.00008

CIS (Center for Internet Security). (2021). CIS Critical Security Controls. https://www.cisecurity.org/controls

Dragoni, N., Ghezzi, C., Lafuente, A. L., y Mazzara, M. (2017). Microservices: Yesterday, Today, and Tomorrow. In Present and Ulterior Software Engineering (pp. 195–216). Springer, Cham. https://doi.org/10.1007/978-3-319-67425-4_12

Erich, F., Amrit, C., y Daneva, M. (2017). A Qualitative Study of DevOps Usage in Practice. Journal of Software: Evolution and Process, 29(6), e1885. https://doi.org/10.1002/smr.1885

Ghali, D., y Guban, M. (2021). Recent trends in web application security threats and countermeasures. In 2021 IEEE 7th International Conference on Applied System Innovation (ICASI) (pp. 336–339). IEEE. https://doi.org/10.1109/ICASI52957.2021.9433884

Gómez, D. (22 de Enero de 2022). Aplicación sobre arquitectura de microservicios en CI desplegada en la nube con seguridad basada en DevSecOps. Obtenido de http://hdl.handle.net/10609/138727

Guitarra, Z. M. (28 de Noviembre de 2022). Creación de una aplicación Backend del módulo de transferencias (cuentas y montos), con microservicios Graphql y Rest en contenedores Docker, para fomentar el levantamiento de una arquitectura Devops. Obtenido de https://repositorio.utn.edu.ec/handle/123456789/13293

Islam, M. D. S., y Akter, S. (2021). Adopting DevSecOps in Microservices-based Software Development: A Systematic Literature Review. In 2021 28th Asia-Pacific Software Engineering Conference (APSEC) (pp. 244–251). IEEE. https://doi.org/10.1109/APSEC53969.2021.00036

ISO. (2018). ISO/IEC 27034-1:2018—Information technology—Security techniques—Application security. International Organization for Standardization.

Janca, T. (2021). Alice and Bob Learn Application Security. En Securing Modern Applications and Systems (págs. 167-191).

Kalliamvakou, E., Bird, C., Zimmermann, T., Begel, A., DeLine, R., y German, D. M. (2014). The promises and perils of mining GitHub. In 2014 11th Working Conference on Mining Software Repositories (pp. 92–101). ACM/IEEE. https://doi.org/10.1145/2597073.2597074

Kim, G., Humble, J., Debois, P., y Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press.

Lacalle, I. (Septiembre de 2022). Diseño y desarrollo de una arquitectura de Internet de las Cosas de nueva generación orientada al cálculo y predicción de índices compuestos aplicada en entornos reales. Obtenido de http://hdl.handle.net/10251/190634

Ley de Instituciones del Sector Financiero (LISF). (2008). Registro Oficial N.º 332, 12 de septiembre.

Ley Orgánica de Protección de Datos Personales (LOPDP). (2021). Registro Oficial N.º 459, 26 de mayo.

Ma, Y., Yan, Z., Du, X., y Chen, W. (2020). Security approaches in container orchestration tools for microservices-based cloud applications: A survey. IEEE Access, 8, 24746–24766. https://doi.org/10.1109/ACCESS.2020.2970243

Mahalakshmi, P., y Magesh, R. (2020). A Security Model for Microservices Architecture. En 2020 5th International Conference on Communication and Electronics Systems (ICCES) (pp. 1072–1077). IEEE. https://doi.org/10.1109/ICCES48766.2020.9137932

Manuj, S., Singh, P., y Dhaliwal, S. K. (2021). SonarQube: A powerful tool for code review and software quality management. International Journal of Emerging Technologies in Engineering Research, 9(8), 456–462.

McZara, J., Kafle, S., y Shin, D. (2020). Modeling and Analysis of Dependencies between Microservices in DevSecOps. https://doi.org/10.1109/SmartCloud49737.2020.00034

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239).

Mohan, S., y Yapa, S. (2019). Exploring the impact of cybersecurity breaches in the banking sector: A data-driven approach. In 2019 3rd International Conference on Cloud Computing Technologies and Applications (CloudTech) (pp. 45–51). IEEE. https://doi.org/10.1109/CloudTech.2019.00015

Mujawar, A. A., y Patel, D. (2021). DevSecOps: Integrating security in DevOps pipeline. International Journal of Emerging Trends in Engineering Research, 9(6), 806–813. https://doi.org/10.30534/ijeter/2021/07962021

Newman, S. (2021). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.

NIST. (2020). Security and Privacy Controls for Information Systems and Organizations (SP 800-53 Rev. 5). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-53r5

OWASP Foundation. (2021). OWASP SAMM 2.0. https://owasp.org/www-project-samm/

OWASP. (2023). OWASP Top Ten Web Application Security Risks. Recuperado de https://owasp.org/www-project-top-ten/

Padrón, J. J. (Junio de 2021). DevSecOps: integración de la seguridad en entornos CI/CD.

Paul, A., Manoj, R., y S, U. (2024). Amazon Web Services Cloud Compliance Automation with Open Policy Agent. https://doi.org/10.1109/ICOECA62351.2024.00063

Pham, R., Pham, P., y Pham, L. (2020). An empirical study of software engineering practices on GitHub. In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 226–233). IEEE. https://doi.org/10.1109/QRS-C51114.2020.00046

Rahman, S. M. A., y Williams, L. (2019). Software security in DevOps: Synthesizing practitioners' perceptions and practices. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 289–298). IEEE. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Rajapakse, R., Zahedi, M., Babar, M., y Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. arXiv preprint. https://arxiv.org/abs/2103.08266

Rajapakse, R., Zahedi, M., Babar, M., y Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. Recuperado de https://arxiv.org/abs/2103.08266.

Rhazili, S. (1 de Febrero de 2023). Securización de microservicios desplegados con Docker y orquestados con Kubernetes. Obtenido de http://hdl.handle.net/10045/131638

Risco, J. A., y Molina, C. W. (16 de Noviembre de 2023). Implementación del modelo DevSecOps en una arquitectura de microservicios on-premise para una empresa del sector bancario. Obtenido de http://hdl.handle.net/10757/671909

Rose, S., Borchert, O., Mitchell, S., y Connelly, S. (2020). Zero trust architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207

Salam, M. M., Basalamah, S., y Jamas, D. N. (2021). A Review of Cybersecurity Threats in the Banking Industry and Proposed Solutions. In 2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET) (pp. 1–6). IEEE. https://doi.org/10.1109/IICAIET51655.2021.9495144

Secretaría Nacional de Planificación . (2024). Plan de Desarrollo para el Nuevo Ecuador 2024-2025. Obtenido de https://www.planificacion.gob.ec/plan-de-desarrollo-para- el-nuevo-ecuador-2024-2025/

Shahin, M. y Babar, M. A. (2021). From continuous integration to continuous security: Leveraging DevSecOps practices. ACM Computing Surveys, 53(3), 1–29. https://doi.org/10.1145/3360006

Shahin, M., Babar, M., Zhu, L. (2019). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943. Recuperado de https://ieeexplore.ieee.org/document/7886930.

Shi, X., Shen, Z., y Huang, T. (2019). Analysis of Maven project management in Java programming based on big data. In 2019 IEEE 5th International Conference on Big Data Security on Cloud (BigDataSecurity) (pp. 255–258). IEEE. https://doi.org/10.1109/BigDataSecurity.2019.00049

Skyone Solutions. (2023). Análisis de vulnerabilidad de la nube. Recuperado de https://skyone.solutions/es/blog/analisis-de-vulnerabilidad-de-la-nube/

Snyk. (2023). Snyk Documentation. https://docs.snyk.io

Sojan, A., Rajan, R., y Kuvaja, P. (2021). Monitoring solution for cloud-native DevSecOps. https://doi.org/10.1109/SmartCloud52277.2021.00029

Sridharan, S., Perera, S., y Jayaratna, A. (2020). A systematic review on continuous integration and continuous delivery/continuous deployment practices in DevOps. IEEE Access, 8, 188840–188866. https://doi.org/10.1109/ACCESS.2020.3031143

Stenzel, K., Braun, E., Weininger, M., y Matthes, F. (2021). Understanding Observability in Microservice-based Environments: A Systematic Mapping Study. En 2021 IEEE 18th International Conference on Software Architecture (ICSA) (pp. 131–140). IEEE. https://doi.org/10.1109/ICSA51549.2021.00022

Suratekar, T., y Dhage, S. (2022). Observability in the DevOps era: A systematic approach with Prometheus and Grafana. IEEE Access, 10, 55410–55419. https://doi.org/10.1109/ACCESS.2022.3175376

Varela, B. (28 de Diciembre de 2021). Desarrollo seguro de software bajo la metodología DevSecOps. Obtenido de http://hdl.handle.net/10609/138449

Ventura, C., Matiolli, G., y Endo, P. T. (2021). DevSecOps and regulations: Analyzing security compliance in CI/CD pipelines. In 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA) (pp. 1–8). IEEE. https://doi.org/10.1109/NCA53616.2021.9688174

Zhang, Y., Liao, G., y Wang, X. (2022). Security challenges in microservice-based systems: A comprehensive survey. IEEE Access, 10, 54684–54699. https://doi.org/10.1109/ACCESS.2022.3174477

7 ANEXOS

Anexo 1: Guía Entrevistas Devsecops

Objetivo: Obtener percepciones detalladas de expertos en desarrollo de software y seguridad informática sobre la implementación de prácticas DevSecOps, el uso de herramientas open source orientadas a microservicios, los retos y beneficios de estas prácticas en el contexto bancario de Quito.

Perfil de los Entrevistados

- **Profesionales objetivo**:
    - Desarrolladores senior.
    - Ingenieros DevOps.
    - Especialistas en seguridad informática.
    - Líderes de proyectos en instituciones bancarias o afines.
- **Criterios de selección**:
    - Experiencia mínima de 2 años trabajando con DevSecOps, herramientas de código abierto, o microservicios.
    - Participación en proyectos relacionados con software bancario.

1. **Preguntas Generales**
    - Por favor, cuéntenos sobre su rol actual y experiencia en el área de desarrollo de software y seguridad.
    - ¿Ha trabajado con metodologías DevSecOps? Si es así, ¿cuánto tiempo lleva implementándolas?
    - ¿En qué tipo de proyectos ha aplicado estas prácticas?
2. **Preguntas Específicas**

Tema 1: Implementación de DevSecOps

- ¿Cuáles son los mayores desafíos que ha enfrentado al implementar DevSecOps en su organización?
- ¿Cómo han influido las prácticas DevSecOps en la cultura organizacional de su equipo?
- ¿Qué cambios operativos o técnicos fueron necesarios para adoptar DevSecOps?

Tema 2: Herramientas Open Source

- ¿Qué herramientas de código abierto utiliza o recomienda para:

1. Análisis de vulnerabilidades (e.g., Trivy, OWASP Dependency-Check)?
2. CI/CD (e.g., Jenkins, GitHub Actions)?
3. Monitoreo y operaciones (e.g., Grafana, Zabbix)?

- ¿Cómo elige las herramientas adecuadas para su entorno de trabajo?
- ¿Ha enfrentado problemas con falsos positivos en las herramientas de seguridad? Si es así, ¿cómo los maneja?

Tema 3: Impacto en la Seguridad

- Desde su experiencia, ¿qué tan efectivas son las prácticas DevSecOps para reducir vulnerabilidades en entornos de microservicios?
- ¿Cómo evalúa el impacto de DevSecOps en la seguridad de las aplicaciones bancarias?
- ¿Considera que DevSecOps mejora los tiempos de respuesta frente a amenazas?

Tema 4: Recomendaciones

- ¿Qué buenas prácticas recomienda para organizaciones que desean adoptar DevSecOps?
- ¿Cuáles son las áreas clave que requieren más atención al integrar herramientas open source?
- ¿Qué capacitaciones o recursos considera esenciales para que los equipos adopten DevSecOps con éxito?

Anexo 2: Encuesta Metodología de desarrollo de software que integra

Tema: Metodología de desarrollo de software que integra:" desarrollo" (Dev), "seguridad" (Sec) y "Operaciones" (Ops) **(DEVSECOPS) CON HERRAMIENTAS OPENSOURCE ORIENTADOS A MICROSERVICIOS PARA LA DETECCIÓN DE VULNERABILIDADES EN EL CICLO DE DESARROLLO DE SOFTWARE.**

Objetivo: Recopilar información sobre la implementación de prácticas DevSecOps con herramientas open source orientados a microservicios para la detección de vulnerabilidades en el ciclo de desarrollo de software en la ciudad de Quito, Ecuador. Con la finalidad de realizar el Trabajo de Grado para la obtención del Título de "**Magíster en Computación con Mención en Seguridad Informática**".

Instrucciones:

Encuesta dirigida a desarrolladores y profesionales de seguridad de la información que trabajen en el área de desarrollo de software en la ciudad de Quito, Ecuador.

Su opinión es muy valiosa para nosotros, por favor responder a las siguientes preguntas basándose en su experiencia. La información será utilizada exclusivamente con fines de investigación académica.

El tiempo promedio para responder la encuesta es de 5 minutos.

Información General:

Lea detenidamente cada pregunta y marque con una (**X**), donde crea conveniente.

Nro.	**Pregunta**	**1**	**2**	**3**	**4**	**5**
1.-	¿A qué sector pertenece la institución en la que trabaja?					

Respuesta: 1= Banca Pública; **2**= Banca Privada; **3**= Cooperativas; **4=** Académico; **5=** Otro.

Nro.	**Pregunta**	**1**	**2**	**3**	**4**	**5**
2.-	¿Cuál es su rol en la institución?					

Respuesta: 1= Desarrollador; **2**= Ingeniero DevOps; **3**= Especialista en seguridad; **4=** Estudiante; **5=** Otro.

Nro.	**Pregunta**	**1**	**2**	**3**	**4**	**5**
3.-	¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?					

Respuesta: 1= Muy Alto; **2**= Alto; **3=** Medio; **4=** Bajo; **5=**Muy Bajo.

Nro.	Pregunta	1	2	3	4	5
4.-	¿Qué importancia cree que tiene la implementación de seguridad en las fases del ciclo de vida de desarrollo de software?					

Respuesta: 1= Extremadamente necesario; **2**= Muy Necesario; **3**= Necesario; **4**=Poco necesario; **5**= No es necesario.

Nro.	Pregunta	1	2	3	4	5
5.-	¿En su institución en que grado se aplican prácticas de DevSecOps en las fases del ciclo de vida de desarrollo de software?					

Respuesta: 1= Siempre**; 2**= Frecuentemente**; 3**= A veces**; 4**= Poco**; 5**= Nunca.

Si la respuesta fue 1,2,3,4 continuar con la encuesta:

Nro.	Pregunta	1	2	3	4	5
6.-	¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software de su institución?					

Respuesta: 1= Muy efectivas; **2**= Efectivas; **3**= A veces efectivas; **4**= Poco efectivas; **5**= Nada efectivas.

Nro.	Pregunta		
7.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para la planificación?		
Respuesta:	Open Project	GitHub Dashboard	Notion
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	Pregunta		
8.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para el versionamiento del código?		
Respuesta:	Git	GitHub	GitLab
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	**Pregunta**		
9.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para la compilación del código?		
Respuesta:	Gradle	Apache Maven	Jenkins
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	**Pregunta**		
10.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para testing del código?		
Respuesta:	SonarQube	Selenium	Gremlin
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	**Pregunta**		
11.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para CI/CD (Release/Deploy) del código?		
Respuesta:	Jenkins	GitHub Actions	Circle CI
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	**Pregunta**		
12.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para operaciones de aplicaciones?		
Respuesta:	Docker	ContainerD	Podman
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

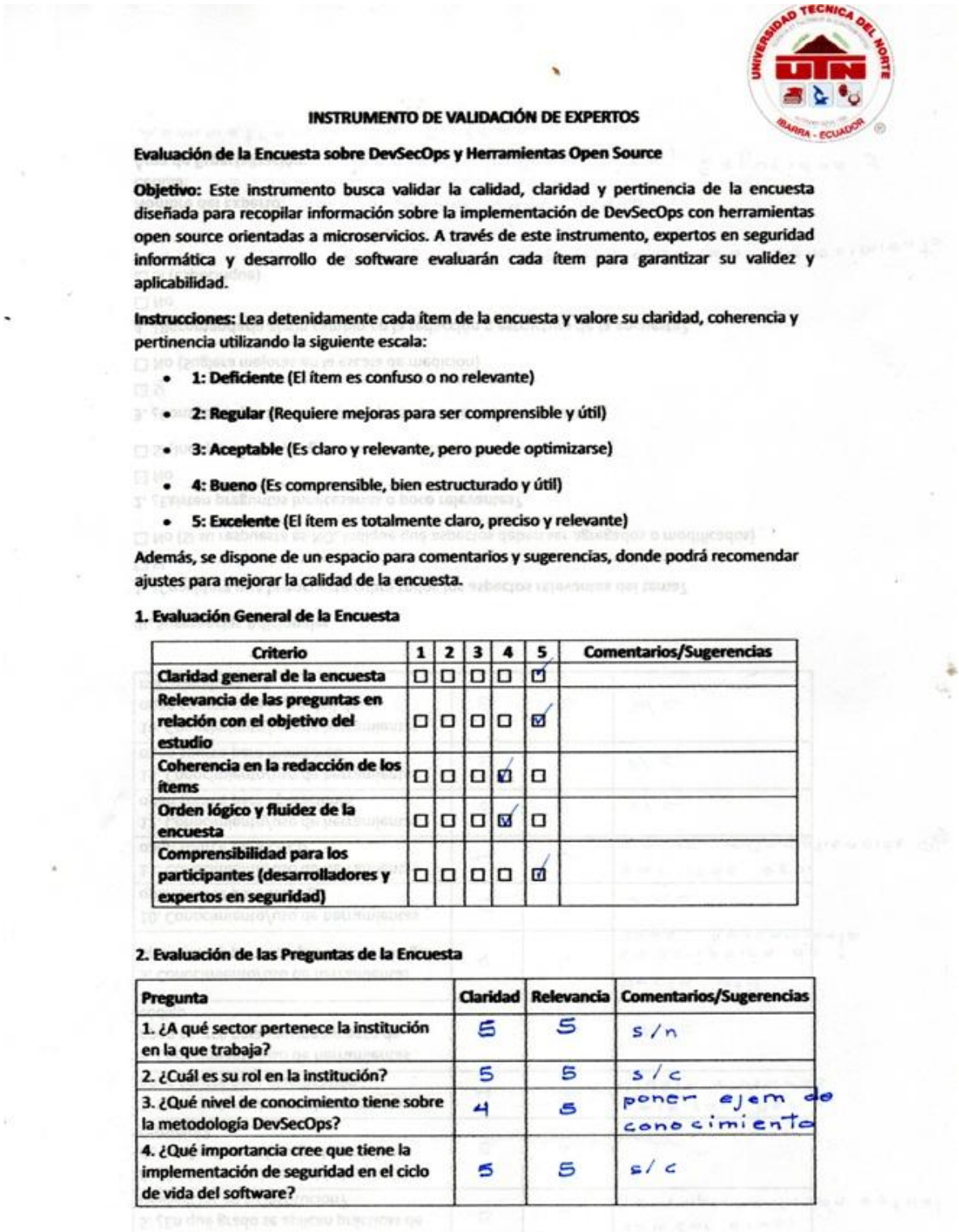
Nro.	**Pregunta**		
13.-	¿En el ciclo de desarrollo de software en qué grado conoce/utiliza estas herramientas open source para monitoreo del código?		
Respuesta:	Kibana	Grafana	Zabbix
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Nro.	**Pregunta**		
14.-	¿En qué grado conoce/utiliza estas herramientas open source para la seguridad de microservicios de su institución?		
Respuesta:	Trivy	OWASP Dependency-Check	Anchore
Muy Alto			
Alto			
Medio			
Bajo			
Nada			

Gracias por su tiempo y colaboración. La información recopilada será tratada con estricta confidencialidad.

Agradecemos su participación. Su aporte es clave para nuestra investigación.

Anexo 3 Validación de Expertos



INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☐	☑	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☐	☑	
Coherencia en la redacción de los ítems	☐	☐	☐	☑	☐	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☑	☐	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	☐	☑	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	5	s/n
2. ¿Cuál es su rol en la institución?	5	5	s/c
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	4	5	poner ejem de conocimiento
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	s/c

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	5	5	colocar nivel de implementación actual
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	4	
7. Conocimiento/uso de herramientas open source para planificación	4	5	colocar herra. mienta + usadas
8. Conocimiento/uso de herramientas open source para versionamiento de código	5	5	S/C
9. Conocimiento/uso de herramientas open source para compilación de código	4	4	Sería útil descripción de cada herramienta
10. Conocimiento/uso de herramientas open source para testing	5	5	S/C
11. Conocimiento/uso de herramientas open source para CI/CD	4	5	ser más esp. de la automatización CI/CD
12. Conocimiento/uso de herramientas open source para operaciones	5	5	S/C
13. Conocimiento/uso de herramientas open source para monitoreo	5	5	S/C
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	S/C

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☑ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☑ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☑ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☐ No

☑ Sí (Especifique)

Para la #3 poner ejm de niveles de conocimiento para mayor claridad.

Nombre del Experto: Marcos Xavier Vera Cedeño

Cédula: 1311993685

Área de Especialización: Ingeniero Informática, Seguridad y Administración de Sistemas

INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☐	☒	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☐	☒	
Coherencia en la redacción de los ítems	☐	☐	☐	☒	☐	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☐	☒	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	☐	☒	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	4	Pregunta clara
2. ¿Cuál es su rol en la institución?	5	5	Adecuada al perfil
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	4	5	Poner ejemplos
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	relevante

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	4	4	Especificar mejor
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	5	Bien enfocada
7. Conocimiento/uso de herramientas open source para planificación	5	5	Específica y estructurada
8. Conocimiento/uso de herramientas open source para versionamiento de código	5	5	alineada al objetivo
9. Conocimiento/uso de herramientas open source para compilación de código	4	4	Mencionan algunas herramientas
10. Conocimiento/uso de herramientas open source para testing	5	5	Importante en seguridad
11. Conocimiento/uso de herramientas open source para CI/CD	4	4	Reformular
12. Conocimiento/uso de herramientas open source para operaciones	5	5	Relevante para entender
13. Conocimiento/uso de herramientas open source para monitoreo	5	5	Específica y fácil de responder
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	Clave para la validación

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☒ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☒ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☒ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☐ No

☒ Sí (Especifique)

En la pregunta 9, sugiero incluir ejemplos de herramientass específicas

Nombre del Experto: David Fernando Torres Zambrano
Cédula: 1718517040
Área de Especialización: ingeniero en electronica y redes De Informaci-

INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☑	☐	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☐	☑	
Coherencia en la redacción de los ítems	☐	☐	☐	☐	☑	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☑	☐	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	☐	☑	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	5	Correcto para segmentar a los participantes
2. ¿Cuál es su rol en la institución?	5	5	Bien planteada
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	4	5	Dar ejemplos de niveles de conocimiento.
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	Muy útil para evaluar la percepción de seguridad en DevSecOps

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	5	5	Pregunta Clara.
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	5	Buena para medir el impacto en la seg. del software
7. Conocimiento/uso de herramientas open source para planificación	4	5	Utilizar ejemplos de herramientas más utilizadas.
8. Conocimiento/uso de herramientas open source para versionamiento de código	5	5	Precisa y alineada con la metodología de la investigación.
9. Conocimiento/uso de herramientas open source para compilación de código	4	4	Se podría integrar en un solo ítem las diferentes herramientas.
10. Conocimiento/uso de herramientas open source para testing	5	5	Específica y bien estructurada
11. Conocimiento/uso de herramientas open source para CI/CD	4	5	Aclarar que es CI/CD.
12. Conocimiento/uso de herramientas open source para operaciones	5	5	Útil en seg de operaciones
13. Conocimiento/uso de herramientas open source para monitoreo	5	5	Bien en el monitoreo.
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	Importante para validar adopción de herramientas.

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☑ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☑ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☑ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☐ No

☑ Sí (Especifique) En la pregunta 7 se sugiere incluir ejemplos de herramientas específicas para mayor precisión en la respuesta.

Nombre del Experto: Jonathan Stalin Delgado Genero
Cédula: 0922651872
Área de Especialización: Magister en Sistemas de la Información General. Magister en Educación Informática, Ingeniero en Sistemas Computacionales.

INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☐	■	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☐	■	
Coherencia en la redacción de los ítems	☐	☐	☐	■	☐	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☐	■	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	■	☐	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	5	
2. ¿Cuál es su rol en la institución?	5	5	
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	4	4	
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	4	5	
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	5	
7. Conocimiento/uso de herramientas open source para planificación	5	5	
8. Conocimiento/uso de herramientas open source para versionamiento de código	4	5	
9. Conocimiento/uso de herramientas open source para compilación de código	5	5	
10. Conocimiento/uso de herramientas open source para testing	5	5	
11. Conocimiento/uso de herramientas open source para CI/CD	4	5	
12. Conocimiento/uso de herramientas open source para operaciones	5	5	
13. Conocimiento/uso de herramientas open source para monitoreo	5	5	
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☑ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☑ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☑ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☐ No

☑ Sí (Especifique) Es mejor agregar opciones de respuesta en las preguntas que hacen referencia a las herramientas DevSecOps.

Nombre del Experto: Pablo Cajas Castillo
Cédula: 1716680374
Área de Especialización: Master Universitario en Seguridad Informática
Ingeniero en Sistemas

INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☒	☐	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☐	☒	
Coherencia en la redacción de los ítems	☐	☐	☐	☐	☒	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☒	☐	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	☒	☐	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	5	Segmenta correctamente a los encuestados
2. ¿Cuál es su rol en la institución?	5	5	Adecuada para conocer el perfil de los participantes.
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	3	4	Se puede mejorar especificando marco de referencia
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	Muy relevante para la adopción de prácticas DevSecOps

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	4	5	Podría incluir una opción de respuesta relacionada con el nivel de adopción
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	4	Sería útil agregar un ejemplo práctico de impacto en la seguridad
7. Conocimiento/uso de herramientas open source para planificación	4	5	Recomendable incluir pregunta sobre herramienta de seguridad
8. Conocimiento/uso de herramientas open source para versionamiento de código	5	5	ok ✓
9. Conocimiento/uso de herramientas open source para compilación de código	3	4	Falta una breve descripción de contexto de uso de estas herramientas
10. Conocimiento/uso de herramientas open source para testing	5	5	Clave para evaluar la seguridad en la fase de pruebas
11. Conocimiento/uso de herramientas open source para CI/CD	4	5	Clarificar si se enfoca en integración o entrega continua
12. Conocimiento/uso de herramientas open source para operaciones	5	5	ok ✓
13. Conocimiento/uso de herramientas open source para monitoreo	4	5	Podría incluirse una herramienta específicas de monitoreo
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	Muy importante en el contexto de microservicios y DevSecOps

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☒ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☒ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☒ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☐ No

☒ Sí (Especifique)

En la pregunta 3 sugiero especificar un marco de referencia de conocimiento para orientar mejor la respuesta.

Nombre del Experto: Diego Xavier Poveda Pilatasig

Cédula: 1722146535

Área de Especialización: Ingeniero en Sistemas

INSTRUMENTO DE VALIDACIÓN DE EXPERTOS

Evaluación de la Encuesta sobre DevSecOps y Herramientas Open Source

Objetivo: Este instrumento busca validar la calidad, claridad y pertinencia de la encuesta diseñada para recopilar información sobre la implementación de DevSecOps con herramientas open source orientadas a microservicios. A través de este instrumento, expertos en seguridad informática y desarrollo de software evaluarán cada ítem para garantizar su validez y aplicabilidad.

Instrucciones: Lea detenidamente cada ítem de la encuesta y valore su claridad, coherencia y pertinencia utilizando la siguiente escala:

- **1: Deficiente** (El ítem es confuso o no relevante)
- **2: Regular** (Requiere mejoras para ser comprensible y útil)
- **3: Aceptable** (Es claro y relevante, pero puede optimizarse)
- **4: Bueno** (Es comprensible, bien estructurado y útil)
- **5: Excelente** (El ítem es totalmente claro, preciso y relevante)

Además, se dispone de un espacio para comentarios y sugerencias, donde podrá recomendar ajustes para mejorar la calidad de la encuesta.

1. Evaluación General de la Encuesta

Criterio	1	2	3	4	5	Comentarios/Sugerencias
Claridad general de la encuesta	☐	☐	☐	☐	☒	
Relevancia de las preguntas en relación con el objetivo del estudio	☐	☐	☐	☒	☐	
Coherencia en la redacción de los ítems	☐	☐	☐	☐	☒	
Orden lógico y fluidez de la encuesta	☐	☐	☐	☒	☐	
Comprensibilidad para los participantes (desarrolladores y expertos en seguridad)	☐	☐	☐	☐	☒	

2. Evaluación de las Preguntas de la Encuesta

Pregunta	Claridad	Relevancia	Comentarios/Sugerencias
1. ¿A qué sector pertenece la institución en la que trabaja?	5	5	Segmentación adecuada para el estudio
2. ¿Cuál es su rol en la institución?	4	5	Incluir opción para roles híbridos
3. ¿Qué nivel de conocimiento tiene sobre la metodología DevSecOps?	4	5	Mejorar los niveles de redacción.
4. ¿Qué importancia cree que tiene la implementación de seguridad en el ciclo de vida del software?	5	5	—

5. ¿En qué grado se aplican prácticas de DevSecOps en su institución?	5	4	Incluir opción para implementación parcial de DevSecOps.
6. ¿Qué impacto han tenido las prácticas de DevSecOps en la seguridad del software?	5	5	
7. Conocimiento/uso de herramientas open source para planificación	4	5	Incluir herramientas comunes.
8. Conocimiento/uso de herramientas open source para versionamiento de código	5	5	
9. Conocimiento/uso de herramientas open source para compilación de código	4	4	Ampliar incluyendo ejemplos de herramientas para compilación de código
10. Conocimiento/uso de herramientas open source para testing	5	5	
11. Conocimiento/uso de herramientas open source para CI/CD	4	5	Evaluar la automatización de CI/CD o su adopción general.
12. Conocimiento/uso de herramientas open source para operaciones	5	5	-
13. Conocimiento/uso de herramientas open source para monitoreo	5	5	-
14. Conocimiento/uso de herramientas open source para seguridad de microservicios	5	5	-

III. Sugerencias Adicionales

1. ¿Considera que la encuesta cubre todos los aspectos relevantes del tema?

☑ Sí

☐ No (Si su respuesta es NO, indique qué aspectos deben ser agregados o modificados)

2. ¿Existen preguntas innecesarias o poco relevantes?

☑ No

☐ Sí (Indique cuáles y sugiera modificaciones)

3. ¿Considera que la escala de respuestas utilizada es adecuada?

☑ Sí

☐ No (Sugiera mejoras en la escala de medición)

4. ¿Recomendaría algún cambio en la redacción o estructura de la encuesta?

☑ No

☐ Sí (Especifique)

En la #9 sugiero incluir herramientas para compilación de código para mayor claridad.

Nombre del Experto: Jonathan Luis López Torres

Cédula: 0930733697

Área de Especialización: Ingeniero en Sistemas Computacionales

Anexo 4 Documentación Detallada de pipeline

.github/workflows

Documentación Detallada del Pipeline

Job: build-and-push

1. **Checkout Repository**

```yaml
- name: Checkout repository
  uses: actions/checkout@v4
  with:
    fetch-depth: 1
```

Propósito: Clona el repositorio en el runner.

- fetch-depth: 1: Solo descarga el último commit para optimizar el tiempo
- Usa la versión 4 de la action para mejor rendimiento y seguridad

2. **Trivy Filesystem Scan**

```yaml
- name: Run Trivy vulnerability scanner (Filesystem scan)
  uses: aquasecurity/trivy-action@master
  with:
    scan-type: 'fs'
    scan-ref: 'webSocketServer'
    format: 'table'
    severity: 'CRITICAL,HIGH'
    output: 'trivy-results.txt'
```

Propósito: Analiza vulnerabilidades en el código fuente.

- scan-type: 'fs': Escaneo del sistema de archivos
- scan-ref: Directorio a escanear
- severity: Solo reporta vulnerabilidades críticas y altas
- format: Salida en formato tabla para mejor legibilidad

3. **Save Trivy Results**

```yaml
- name: Save Trivy scan results
  uses: actions/upload-artifact@v4
  with:
    name: trivy-scan-results
    path: trivy-results.txt
    retention-days: 7
```

Propósito: Guarda los resultados del escaneo como artefacto.

- retention-days: 7: Mantiene los resultados por una semana

- Permite descargar y revisar los resultados posteriormente

4. **GitHub Container Registry Login**

```yaml
- name: Log in to GitHub Container Registry
  uses: docker/login-action@v3
  with:
    registry: ${{ env.REGISTRY }}
    username: ${{ github.actor }}
    password: ${{ secrets.GITHUB_TOKEN }}
```

Propósito: Autentica con GitHub Container Registry.

- Usa las credenciales del workflow automáticamente
- Permite push/pull de imágenes privadas

5. **Extract Docker Metadata**

```yaml
- name: Extract metadata for Docker
  id: meta
  uses: docker/metadata-action@v5
  with:
    images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
    tags: |
      type=sha,prefix=sha-
```

Propósito: Genera metadatos para la imagen Docker.

- Crea tags basados en el SHA del commit
- Genera etiquetas automáticamente
- El ID 'meta' permite referir estos valores en otros steps

6. **Build and Push Docker Image**

```yaml
- name: Build and push Docker image
  uses: docker/build-push-action@v5
  with:
    context: ./webSocketServer
    file: ./webSocketServer/Dockerfile
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
```

Propósito: Construye y publica la imagen Docker.

- context: Directorio base para el build
- file: Ubicación del Dockerfile
- push: true: Publica automáticamente la imagen
- Usa los tags y labels generados en el paso anterior

7. **Trivy Image Scan**

```yaml
- name: Run Trivy vulnerability scanner
  uses: aquasecurity/trivy-action@master
  with:
    image-ref: ${{ steps.meta.outputs.tags }}
    format: 'table'
    output: 'trivy-image-results.txt'
    severity: 'CRITICAL,HIGH'
```

Propósito: Escanea vulnerabilidades en la imagen Docker.

- Analiza la imagen recién construida
- Genera un reporte separado para vulnerabilidades de la imagen

Job: deploy

1. **AWS Credentials Configuration**

```yaml
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ${{ secrets.AWS_REGION }}
```

Propósito: Configura las credenciales de AWS.

- Establece el contexto de AWS para el deployment
- Usa secrets seguros para las credenciales

2. **EC2 Deployment**

```yaml
- name: Deploy to EC2
  uses: appleboy/ssh-action@master
  with:
    host: ${{ secrets.EC2_HOST }}
    username: ${{ secrets.EC2_USERNAME }}
    key: ${{ secrets.EC2_SSH_KEY }}
    script: |
      # Comandos de deployment
```

Propósito: Despliega la aplicación en EC2.

- Ejecuta comandos remotos vía SSH
- Realiza las siguientes operaciones:
    - Login en GitHub Container Registry
    - Obtiene el tag de la imagen más reciente
    - Detiene y elimina el contenedor anterior
    - Despliega el nuevo contenedor

Variables de Entorno Globales

```yaml
env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}
```

Propósito: Define variables globales para el workflow.

- REGISTRY: Define el registro de contenedores a usar
- IMAGE_NAME: Usa el nombre del repositorio automáticamente

Permisos

```yaml
permissions:
  contents: read
  packages: write
  security-events: write
```

Propósito: Define los permisos necesarios para el workflow.

- contents: read: Permite leer el repositorio
- packages: write: Permite publicar paquetes/imágenes
- security-events: write: Permite guardar resultados de seguridad